Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Desenvolupament d'un videojoc

Document: Memòria

Alumne: Sergio Rustarazo Bejarano

Tutor: Antonio Rodríguez Benítez
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i Sistemes Informàtics

Convocatòria (mes/any) Setembre / 2019

Universitat de Girona

EPS UdG Escola Politècnica Superior

BATTLELAND

TREBALL FÍ DE GRAU

DESENVOLUPAMENT D'UN VIDEOJOC

GRAU EN DISSENY I DESENVOLUPAMENT DE VIDEOJOCS

Autor: Sergio Rustarazo Bejarano

Tutor: Antonio Rodríguez Benitez

ÍNDEX

1. INTRODUCCIÓ

Aquest document conté tota la informació sobre el disseny i desenvolupament del videojoc *Batteland* realitzat amb el propòsit de servir com a treball de final de grau. Conté la memòria amb tots els passos que s'han seguit per dur-ho a terme, explicant detingudament cada apartat, els obstacles que han aparegut, les decisions preses, etc.

Abans de començar, per posar una mica en context, explicaré molt breument el concepte del videojoc. A la figura 1 podem veure una captura per posar en context.

Battleland és un videojoc d'estratègia per torns. És un videojoc enfocat al multijugador, i la idea en poques paraules és que dos jugadors s'enfronten en un camp de batalla. Cadascun controla 3 personatges diferents, i el primer jugador que tots els seus personatges siguin eliminats haurà perdut la partida.

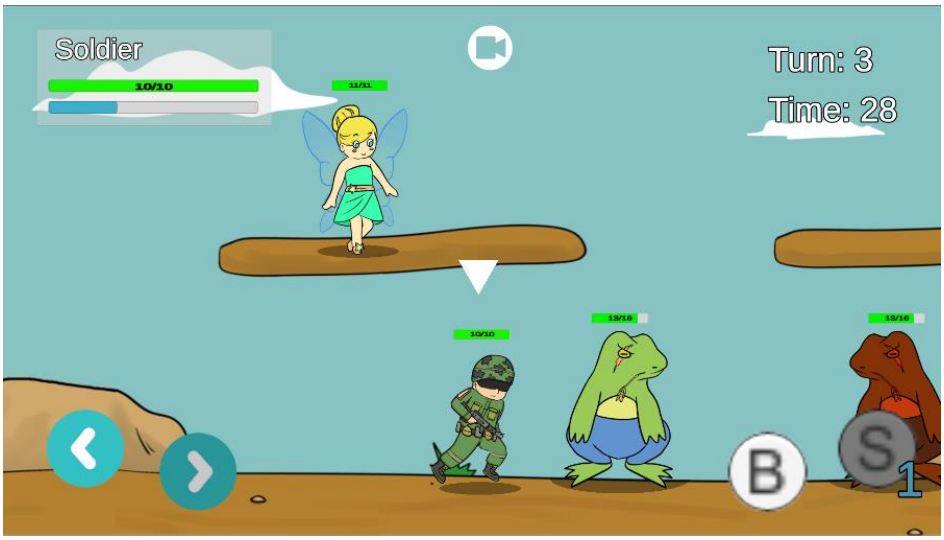


Figura 1 Captura d'una partida de Battleland

Per començar, en aquesta introducció s'explicarà una mica per sobre els motius per a fer aquest treball així com les motivacions per dur-lo a terme, i a continuació es detallaran alguns aspectes relatius al document en si mateix i a com s'han repartit els esforços pel que fa als quatre mòduls més destacables d'un videojoc; tecnologia, mecàniques, art, narrativa.

1.1 MARC DEL PROJECTE

Fa temps que ha deixat de ser una sorpresa que la indústria dels videojocs sigui la indústria de l'entreteniment que genera més diners, fins i tot en certs punts superar per si mateixa al cine i la música junts. Espanya és un dels molts països on passa això, com podem veure a la figura 2 amb les dades recollides per AEVI (Asociación Española de Videojuegos) . El que en un principi semblava que només apuntava a un públic molt concret, ha aconseguit adaptar-se i gràcies a la inesgotable imaginació dels dissenyadors el futur dels videojocs sembla continuarà evolucionant. No ha sigut fàcil pels videojocs arribar a l'estat en el qual es troba, ha hagut de lluitar sobretot contra els prejudicis de la societat.

Figura 2 Comparació de facturació entre Videojocs, Cine i Música

Els videojocs han esdevingut un dels mitjans d'entreteniment més populars del món, i això s'ha aconseguit gràcies a diferents motius.

Si bé en un principi es relacionava a la gent que juga a videojocs amb adolescents antisocials que es tancaven a casa tot el dia, els videojocs han aconseguit canviar aquesta percepció. Primerament l'interès d'arribar a molt més públic va desembocar en multitud de gèneres molt diferents amb el qual és gairebé impossible que ningú no se senti atret, però també han aparegit jocs amb dificultat graduable, jocs molt llargs o jocs on les partides duren 3 minuts. La varietat pel que fa a videojocs ha fet que s'expandeixi molt ràpidament i puguin ser gaudits tant per gent que vol viure una gran història, gent

que vol jugar amb amics, o gent que s'avorreix a l'autobús camí al treball, entre molts d'altres. A la figura 3 podem veure el comentat en aquest paràgraf i com cada vegada més el perfil dels jugadors es va repartint entre més població.

Figura 3 Perfil dels jugadors a Espanya

Seguidament, es va veure que els videojocs podien omplir un espai molt més gran que l'entreteniment. Des del seu ús per l'ensenyament a les escoles, com els cada cop més populars coneguts *serious games*[1] o amb l'ús de la gamificació[2], així com per coses més concretes com aprendre a dibuixar o a tocar un instrument. També han esdevingut importants en molts camps com la medicina i s'utilitzen per simulacions de tota mena, a la figura 4 tenim un parell de captures de demostració d'aquest fenomen.

Figura 4 Captures del serious game "Lissa" i un simulador de vol

[1] Els *serious games* són jocs els quals es seu proposit és la formació i la ensenyança per sobre de l'entreteniment.

[2] La gamificació és l'ús d'elements dels videojocs en activitats o contextos aliens a aquest amb l'objectiu de transferir coneixements fent més amé el procés.

Finalment i un dels motius principals pel qual han aconseguit ser tan importants és el fet que amb el temps i a raó de diferents estratègies, s'ha simplificat molt la manera d'accedir a ells i poder-los gaudir.

Aquest és el tema que es tracta més en aquest document, i és que tot i que veurem que gràcies a aquesta simplificació per accedir al contingut els videojocs han crescut exponencialment, els jugadors es veuran afectats negativament d'una forma o una altra. Amb total seguretat es podria afirmar que actualment tothom té algun dispositiu multimèdia amb el qual pot jugar a videojocs, des de les plataformes específiques com són les consoles fins a elements més comuns com són ordinadors o dispositius mòbils. A continuació veiem que a Espanya mateix, els jugadors es reparteixen entre molts tipus de plataformes.

Figura 5 Dispositius més utilitzats per jugar a Espanya

Tot i això però, i pel mateix motiu pel qual els videojocs s'han consolidat i expandit tant, veiem que hi ha un aspecte negatiu que xoca amb un dels pilars dels videojocs; jugar amb els teus amics. Ara hi ha una base molt més gran de jugadors, la qual encara continua expandint el seu públic apropant-se a persones de diferents edats i gusts, però aquesta base s'ha dividit per culpa de què els videojocs han trobat diversos camins per a assolir l'èxit. Uns han trobat el seu lloc en dispositius tàctils, altres continuen jugant-se amb teclat i ratolí, i altres potser atrauen un gran públic sigui quin sigui el mètode, però per limitacions tecnològiques o empresarials han dividit la seva comunitat.

1.2 MOTIVACIONS

Fer un videojoc és per mi tant el cim de la meva etapa com a estudiant universitari, com el que serà la primera rajola el meu futur com a dissenyador i desenvolupador de videojocs. No hi ha millor manera de posar en pràctica tot l'après durant el grau i demostrar-ho que en el projecte de final de carrera, i alhora serà la meva millor carta de presentació davant el món dels videojocs.

La motivació per a realitzar aquest treball és analitzar el problema comentat al marc del projecte i explorar les solucions possibles. Realitzar un videojoc que no fracturi la seva comunitat i unir-la superant dificultats com que cada jugador estigui executant el seu client en plataformes diferents. Battleland permetrà als jugadors d'Android i Windows jugar partides entre ells siguin de la mateixa plataforma o no.

Figura 6 Logotips d'Android i Windows

Amb això, en aquest treball també s'espera explorar les dificultats de dissenyar un videojoc fora de l'ambient educatiu, és a dir, amb la mentalitat de què un videojoc a final de comptes és un producte, i per tant s'ha de tenir una visió empresarial. Analitzar mercat, tècniques per atraure públic, com mantenir amb vida un joc, etc.

1.3 ELECCIÓ DEL VIDEOJOC

En la ment vaig considerar diferents possibilitats de videojocs, però des d'un principi tenia bastant clar que volia que fos per dispositius mòbils. Com s'ha vist en les dades recollides d'AEVI en l'apartat anterior, és la plataforma on actualment hi ha una base molt important de jugadors i la que està creixent més.

Potser és a causa del fet que sempre feia molts viatges amb autobús que vaig veure que els videojocs de mòbil anaven amb l'objectiu de fer jocs molt simples amb un cicle de vida de 3 dies. Al final en haver tant contingut de videojocs i ser tan fàcil de publicar feia que aquests acabaven sent jocs monòtons i fets amb poc entusiasme.

Amb això vaig decidir que volia fer un joc de partides curtes, però amb suficient profunditat com per a no deixar-lo abandonat al cap d'una setmana. A diferència dels jocs que abunden el mercat actual dels dispositius mòbils, Battleland havia de ser un joc que tingués moltes possibilitats, que cada partida amb una composició diferent pogués representar una experiència completament diferent d'una altra i d'aquesta manera fer un joc amb molta rejugabilitat i no et cansessis ràpidament.

1.3.1 PER QUÈ ENFOCAT AL MULTIJUGADOR?

Els darrers anys els jocs multijugador han sofert un canvi molt important. En un principi el multijugador només es podia jugar de manera local, i amb molt poques persones, però ara els videojocs multijugador solen concentrar unitats massives de persones i els agrupa a tots en una mateixa partida. Això s'ha produït principalment gràcies al fet que la Internet ja està instaurat en tots llocs i amb unes molt bones infraestructures i tecnologies que el suporten.

Tant ha evolucionat el multijugador, que el que abans solia ser una component complementaria al videojoc principal, ha esdevingut en molts casos l'atractiu principal del videojoc, i fins a arribar a actualment on els videojocs basen el seu *gameplay*[3] enfocat únicament al multijugador, sigui competitiu o cooperatiu. Aquesta evolució de la tecnologia i del multijugador també, ha provocat que alguns modes com el de pantalla dividida com es pot veure a la figura 7 s'hagin pràcticament extingit. Simplement i com és natural perquè les coses evolucionen, ara hi ha altres modes millors de gaudir-lo.

Figura 7 Captura d'un joc en mode pantalla dividida

Amb diferents idees sobre la taula, Battleland pensava que podria funcionar bastant bé perquè agafa dos components que s'estan explotant molt els últims anys, com són jocs amb molts personatges diferents i multijugador online. També m'he decantat per aquest pel conflicte plantejat de realitzar un videojoc multiplataforma, perquè penso que és un joc fàcilment adaptable a les dues plataformes seleccionades.

[3] La traducció al nostre idioma del terme *Gameplay* seria jugabilitat. En el disseny de videojoc s'utilitza per explicar les regles del joc i el seu funcionament.

1.3.2 PER QUÈ MULTIPLATAFORMA?

Aquest joc podria haver sigut fàcilment pensat per una sola plataforma, però a part de dissenyar un videojoc volia que aquest treball també em servis per analitzar aquest problema mencionat anteriorment i analitzar el perquè s'ha arribat a aquesta situació i de quines formes es podria resoldre, i finalment per al meu videojoc implementar-lo de forma que pugui jugar amb altres persones encara que no sigui en el mateix dispositiu.

1.4 PROPÒSIT I OBJECTIUS DEL PROJECTE

L'objectiu principal de realitzar aquest treball és el de realitzar un videojoc, seguint tots els passos necessaris com si és tractes d'un desenvolupament professional.
Es vol abastar tots els camps possibles i dissenyar-lo pensant en una estratègia de què el videojoc funcioni i que l'experiència se senti per igual sent multiplataforma.
Finalment crear una petita versió del joc o prototip que en un futur es pogués reprendre i arribar a realitzar un videojoc complet i posteriorment publicar-lo perquè pugui ser gaudit pels jugadors.
Per assolir aquest objectiu s'han marcat els següents objectius específics:

- Explorar el mercat i definir un públic
- Analitzar la tecnologia
- Definir el guió del joc
- Definir l'abast del joc
- Dissenyar i implementar el joc local
- Dissenyar i implementar el joc online
- Fer-lo multiplataforma
- Fer un correcte balanceig del joc

Amb aquests objectius també aprendré moltes coses que encara no havia tocat mai, o només havia tractat teòricament.

1.5 QUADRE D'AUTOVALORACIÓ

En ser un videojoc complet, tots els aspectes tenen la seva part de representació en el resultat final.

Normalment un videojoc és creat per un equip multidisciplinari, però en aquest cas m'encarregaré de totes les diferents parts que l'engloben. A continuació es mostra un quadre (Taula 1) amb una valoració personal de quan esforç i valor es dedica a cada un d'aquests quatre elements.

Estètica	20 %
Narrativa	5%
Mecàniques	40 %
Tecnologia	35 %

Taula 1 Quadre d'autovaloració

Els dos blocs on es gastarà més recursos per aquest videojoc són les mecàniques i tecnologia, els quals he decidit que comparteixin un 65% del projecte. Per a les mecàniques he decidit apostar gran part de l'atractiu del joc, sent l'element de *hook* [4] envers els jugadors.

Tot i que sobre el paper és un joc simple, s'apostarà per una gran quantitat de mecàniques diferents que faran que els jugadors s'interessin per explorar totes les possibilitats. Pel que fa a la tecnologia, té un percentatge elevat bàsicament pel punt que volem tractar en aquest videojoc sobre fer-lo multiplataforma. També perquè el joc està enfocat al multijugador i s'implementarà Online.

Seguint l'ordre de major a menor, trobem l'estètica. L'estètica es un factor diferencial. Pot ser molt bon joc però si no entra pels ulls, no aconseguirà plaça en la industria, ja que és el primer que veiem i uns bons gràfics per si mateixos ja poden atraure un gran públic desinteressat. En haver-hi diferents personatges i escenaris se li haurà de dedicar temps.

[4] En el món dels videojocs s'utilitza la paraula *hook* per descriure la component del videojoc dissenyada per atraure l'atenció dels jugadors.

Finalment com es un joc enfocat al multijugador competitiu la narrativa no té gaire rellevància. Tot i així es vol crear un món i uns personatges amb transfons pels jugadors més curiosos, i que doni suport alhora de dissenyar mecàniques.

1.6 ORGANITZACIÓ DEL DOCUMENT

Per fer aquest document se segueix una guia que ha sigut facilitada per la universitat.

1. Introducció, motivacions, propòsit i objectius del projecte i distribució de tasques
2. Estudi de viabilitat
3. Planificació
4. Marc de treball i conceptes previs
5. Disseny del videojoc
6. Implementació i proves
7. Resultats
8. Conclusions
9. Treball futur
10. Bibliografia
11. Annexos
12. Manual d'usuari i d'instal·lació

ESTUDI DE VIABILITAT

Abans de posar en marxa un projecte d'aquest tipus, és important valorar si la idea és rendible. Els videojocs generalment suposen una inversió inicial molt gran, tant en el mateix desenvolupament com en la campanya de màrqueting. És per aquest motiu que és molt important fer un correcte estudi de viabilitat abans de començar.

En aquest punt, per fer un correcte estudi de viabilitat sobre el nostre videojoc es tractaran diferents temes que juntament ens poden donar una idea de com funcionarà el nostre videojoc una vegada en el mercat. S'utilitzaran valors orientatius en moltes situacions, ja que la inexperiència en aquests temes pot portar a petites variacions de com seria en realitat.

Una de les tasques importants en aquesta fase serà identificar el nostre jugador objectiu, perquè posteriorment haurem de treballar sobre aquest en dos aspectes, com entretenir-lo i com empatitzar amb ell.

Tot i que el videojoc òbviament està fet per una sola persona en el seu temps lliure i sense cap inversió, es tractarà com un desenvolupament professional amb tot el que significa, és a dir estimarem costos i recursos humans com si es tractes d'un projecte d'una petita empresa de videojocs.

2.1 RECURSOS NECESSARIS

2.1.1 EINES

En aquest apartat se citaran les eines utilitzades per a realitzar el TFG en la seva totalitat i una petita descripció de cada una d'elles.

- **Ordinador Lenovo Ideapad 700**: s'utilitzarà com a eina principal per desenvolupar el videojoc i també com a dispositiu de proves.
- **Dispositiu Android Huawei Mate 10 Lite**: S'utilitzarà principalment com a eina de proves.
- **Microsoft Word**: editor de textos de la família Microsoft que s'utilitza per a la redacció d'aquest informe mateix.
- **Unity v2018.3**: motor de joc que s'utilitza per al desenvolupament.
- **Clip Studio paint**: software de dibuix per a realitzar totes les coses en referència a l'art del joc.
- **Wacom Intuos Comic**: tauleta gràfica per a realitzar els diferents *assets* artístics.
- **Photoshop**: eina professional per a retoc d'imatges.
- **Trello**: Servei web de planificació de tasques.

2.1.2 RECURSOS HUMANS

Com he mencionat aquest joc està completament produït per una sola persona amb fins educatius, però en aquest treball es tractarà com un servei professional i es quantificarà les hores de treball per calcular quants diners es necessitarien per pagar l'equip de desenvolupament.

Dividirem el mateix treball d'una persona en 3 diferents per així poder valorar també altres aspectes com a quines seccions s'ha dedicat més temps i poder-ne fer una opinió.

- Dissenyador principal i director de joc
- Desenvolupador principal i *tester*[5]
- Artista principal i guionista

[5] Es diu *tester* a la persona encarregada de provar el videojoc per detectar errors

2.2 PRESSUPOSTOS INICIALS

En aquest apartat valorarem els pressupostos que ens farien falta per desenvolupar el projecte. Òbviament, aquests valors són orientatius i poden canviar molt del que seria el resultat final. Cal remarcar que pel projecte no s'ha invertit diners específicament, sinó que s'ha utilitzat eines de les quals ja es disposava.

1.2.1 EINES UTILITZADES

Els preus que apareixen aquí no són els que valien els productes en el moment de la seva compra, però es considera que és material comprat per a realitzar el desenvolupament i per tant el preu és el que tenen en el moment actual a les tendes.

Eina	Cost
Microsoft Word	135,00 €
Wacom Intuos Comic	59,99 €
Clip Studio Paint	49,99 €
Ordinador Lenovo Y700	845,59 €
Huawei Mate 10 Lite	149,99 €
Photoshop	290,17 €
TOTAL	**1530,73**

Taula 2 Pressupost eines

Si suposéssim que els tres treballadors tinguessin a la seva disposició els mateixos recursos, el preu total sortiria a 4592,19 €

1.2.2 RECURSOS HUMANS

Aquests números que es pressenten a continuació es basen en les dades recollides per infojobs a Espanya el 28 de novembre de 2018 com a estudi dels últims 5 anys. No són exactament precises perquè he fet que una persona faci més d'una funció, ja que en petites empreses és com se sol treballar i es el que es vol simular amb aquest projecte.

Treballador	**Cost/hora**
Dissenyador principal i director de joc	16,6 € / hora
Desenvolupador principal i tester	18,3 € / hora
Artista principal i guionista	16,6 € / hora
TOTAL	51,5 € / hora

Taula 3 Pressupost equip

Són uns números ficticis perquè realment tot el projecte és desenvolupat per una persona, però aquest anàlisi servirà per començar a conèixer el sector. Una vegada feta la planificació i calculat el nombre d'hores podem mesurar el cost total hipotètic del projecte.

2.3 ESTUDI DE MERCAT

Abans d'arriscar-nos a invertir en el nostre videojoc, és necessari estudiar la competència i consultar el mercat, ja que no hem de caure en el pensament de què el que a nosaltres ens sembla una idea molt bona serà el mateix que pensi el públic.

En aquest apartat, entre d'altres mirarem de valorar diferents qüestions com veure per a quin públic dissenyarem el nostre producte i perquè aquest públic ens escollirà. En resum, és una forma de saber la resposta que obtindrem del mercat davant el nostre producte.

S'analitzen diferents coses, oferta i demanda, així com els preus, etc. D'aquesta forma poden definir més el nostre producte, i coneixerem de forma exhaustiva els principals competidors.

Entre altres coses, una mica per sobre les més importants a valorar a l'hora de decidir a on volem publicar el nostre videojoc són:

- Plataformes més populars
- On juga el públic a qui l'atrau més el nostre tipus de joc

Les dades recollides per AEVI a la introducció són una base molt important per començar a estudiar la viabilitat del projecte.

Llavors hem de calcular, ja que potser el nostre públic juga més en dispositius portàtils, però hi ha un percentatge més gran de jugadors amb ordinador. Hem de fer les diferents consideracions per prendre la decisió final.

2.3.1 MODEL DE NEGOCI

Inicialment hem d'estudiar el model de negoci que funcionarà millor amb el nostre joc, és molt important perquè d'aquí és on veurem si realment és viable la nostra idea. A continuació es mostra una llista de possibilitats de model de negoci que he pogut extreure dels apunts del grau i al final es fa un petit pensament per decidir quina és la millor possible per Battleland.

Business Models

In-Game Advertising
Around-Game Advertising
Finder's Fee from First Dollar
Advert-Games/Advergaming/Re-Dressed Games
Try before You Buy
Episodic Entertainment/Expansion Packs
Buy the Win
Insurance
Financing
Velvet Rope or Member's Club
Subscription
Support Tiers
Become a "Brand Member"
In-Game Stores and Microtransactions
Selling Consumables
Skill-Based Progressive Jackpots
Player-to-Player Wagering and Item Sales/Trades
Pay Players to Meet a Challenge
Charityware
Sponsored Games/Donationware
Pay per Play/Pay as You Go/Pay for Time
Player-to-Player Trading/Auctions
Foreign Distribution Deals
Sell Player Access/Co-Registration Offers
Freeware
Loss Leaders
Peripheral Enticement
User-Generated Content
Pay for Storage Space
Host a Private Game Server
Rentals
Licensing
Sell Branded Physical Items
Pre-Sell a Game to Its Players
Before-Game Advertising
Virtual Item Sponsorship
Add Download Insurance
Feed Me or I Die!
Methods of Avoiding Buyer's Regret

Com veiem hi ha moltes possibilitats, però principalment i com veurem més endavant estudiant la competència bàsicament els models de negoci més populars són:

- ***Free to play (joc "gratuït")**:*
    - *In-Game stores and microtransactions* (micro-pagaments per contingut a dins del joc)
    - *In-game advertising* (anuncis patrocinats a dins del joc)
    - *Demo* (et deixen probar una part petita del joc, però si vols continuar jugant has de pagar)

- ***Pay per play*** **(el més tradicional, comprar el joc)**

En un principi, a Battleland interessaria que fos un joc *Free to Play* amb microtransaccions. Sent gratuït podem arribar a molt públic i llavors deixar en les seves mans si volen gastar diners o no, però sense limitar el contingut ni creant-los desavantatges envers els jugadors que posen diners, perquè si no es torna un joc desequilibrat i frustrant.

2.3.2 ESTAT DE L'ART

El videojoc ha de aconseguir fer-se un lloc en el mercat, el qual és bastant complicat tenint en compte la magnitud de videojocs que ja hi estan bastant acomodats i tenen una bona base de jugadors fidels. Per començar llavors, hem de fer un estudi dels videojocs que actualment estan funcionant millor i estudiar el perquè.

És important agafar els millors exponents del gènere i veure que fan bé i malament, com podem millorar-lo o aportar alguna cosa nova que ens diferenciï dels altres.

Aquí primerament farem una cerca dels exponents del gènere i farem una breu descripció, llavors farem un quadre comparatiu amb els videojocs seleccionats i ponderarem uns apartats concrets que es considera més decisius en un videojoc. Per realitzar aquesta recerca de videojocs s'ha utilitzat les paraules clau: *worms game alternatives* al cercador Chrome i s'ha cercat a la tenda Play Store també amb la paraula: *Worms*. Com els cercadors són intel·ligents, són capaços d'associar la paraula amb els diferents jocs semblants que és el que es buscava.

Worms (saga)

Figura 8 Captura d'un joc de la saga Worms

Worms és la inspiració en la qual es basa Battleland.

És més bé una saga que un títol, ja que *Worms* ha tingut moltes versions i sempre ha sigut un videojoc molt aclamat i prestigiós. Per exemple, una de les seves moltes versions és la de Facebook, que és la que es veu en la Figura 8.

En aquests jocs s'enfronten dos o més jugadors que controlen un o diversos personatges durant cert temps, amb l'objectiu d'eliminar els personatges dels adversaris. En aquest cas, els personatges són representats per cucs (*Worms* significa "cucs", en anglès) en una illa que sura en una gran massa d'aigua amb un ambient caricaturesc, comptant amb animació humorística de dibuixos animats i un variat arsenal d'armes fictícies.

Warlings

Figura 9 Captura del joc Warlings

Warlings segueix les petjades del que es vol fer amb Batteland. Agafa un joc com a referència i li atorga algun canvi que li dóna una nova personalitat al videojoc. Com veiem a la figura 9, és molt similar a *Worms* però canviant els cucs per un altre tipus de personatges.

Warlings: Armageddon és un joc d'estratègia per torns en el qual els jugadors podran controlar un petit 'exèrcit' d'entre tres i cinc simpàtics personatges, amb els quals s'hauran d'enfrontar a altres tants personatges en un escenari tancat en dues dimensions.

La mecànica del joc és pràcticament idèntica a la del clàssic *Worms*. És a dir, per torns cada jugador podrà moure a una de les seves criatures, amb la qual haurà d'intentar atacar causant el màxim dany possible als seus enemics. La partida s'acabarà quan un dels dos bàndols es quedi sense reclutes.

Territory War

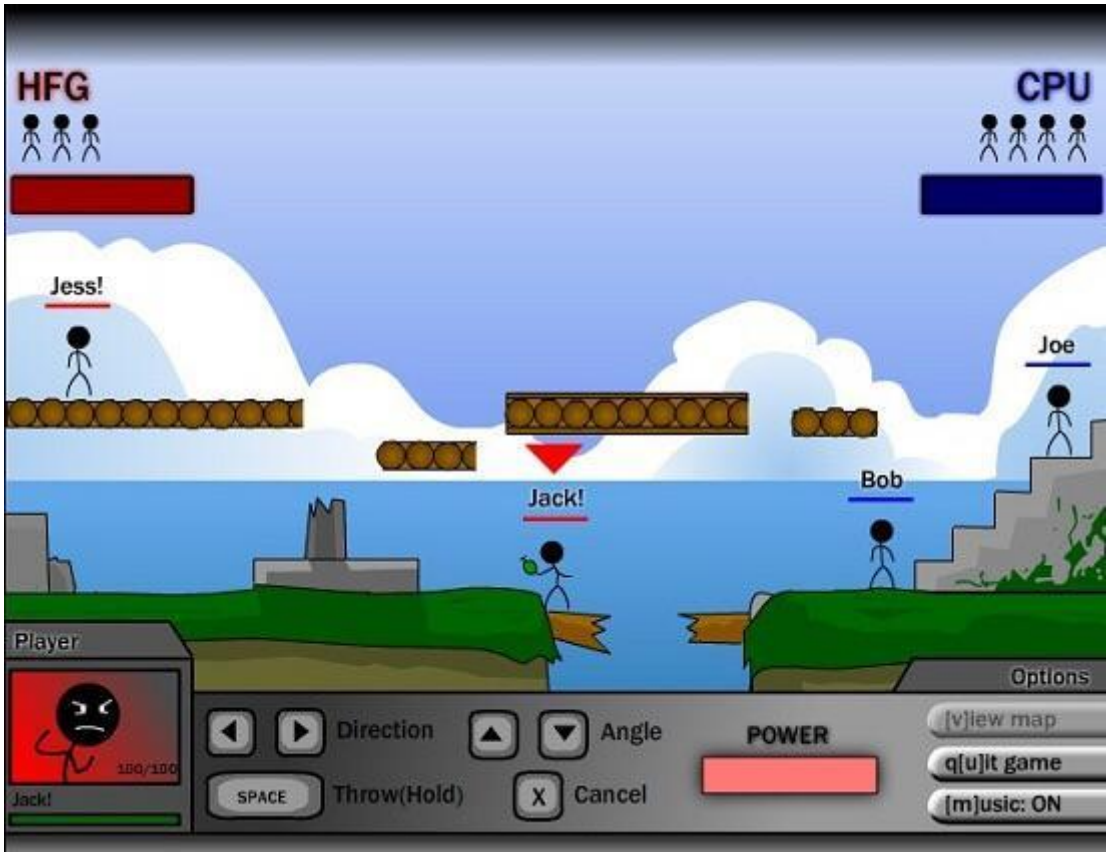


Figura 10 Captura del joc Territory War

Un altre joc a l'estil Worms com podem veure a la figura 10. *Territory war* és un joc més antic però té la mateixa essència. En aquest cas, tenia un mode arcade contra la màquina en el qual anaves superant els nivells a mesura que derrotaves diferents exèrcits enemics. Entre d'altres característiques menys importants, es podia posar noms als personatges. Battleland també sorgeix una mica d'aquest joc, perquè va ser la primera presa de contacte amb el gènere.

Quadre comparatiu

A la taula número 4 analitzem els videojocs vistos en aquest apartat i extreiem les caracteristiques més importants, tot seguit ho comparem amb el que oferirà Battleland.

Nom	Plataforma	Idioma	Model de negoci	Joc creuat
Worms	Multiplataforma	Multidioma	Pay to play (uns 5 €)	No
Warlings	Android	Anglès	Free to play (in-game advertising)	No
Territory war	PC (es necessita adobe flash)	Anglès	Free to play (around-game advertising)	No
Battleland	PC i Android	Anglès	Free to play (In-Game stores and microtransactions)	Si

Taula 4 Quadre comparatiu

2.4 PÚBLIC OBJECTIU I PERFIL DEL JUGADOR

Una vegada hem analitzat el mercat i hem vist sobre quin rang de jugadors ens mourem, hem d'aprofundir per veure com hem de dissenyar el joc per tal d'atraure'l.

Hem de tenir clares algunes coses que poden ser difícils d'encarar per un dissenyador:

- **No ets el jugador**: no hem de pensar que el jugador serà com nosaltres. Ens hem d'imaginar que pot ser molt diferent i per tant hem d'analitzar diferents maneres perquè el jugador se senti còmode jugant el nostre joc.
- **El jugador no és el teu oponent**: no lluitem contra el jugador. En dissenyar hem de pensar a entretenir el jugador, no fer-li perdre els nervis. Hi ha moltes maneres de fer això i és la funció del dissenyador pensar la millor manera.
- **Pensa en el jugador primer**: no pots començar a dissenyar res del joc sense primer fer-te la pregunta de quina és la funció del jugador i què farà aquest quan tingui el joc entre les mans.

Tipologia de jugadors. Descripció del perfil del jugador del joc

Per definir el nostre perfil de jugador del joc, ens basarem en la tipologia de jugadors definida per Richard Burtle i que hem estudiat durant el curs.

Segons Richard Burtle, podem classificar l'estil de joc dels jugadors en 4 diferents tipus de perfil.

- ***Achievers***: tenen com a objectiu resoldre el major número de reptes i dificultats. Obtenen el plaer resolent situacions complexes. Interessats a actuar en el món.
- ***Explorers***: els agrada explorar a fons el videojoc, descobrir els seus secrets i aprendre coses desconegudes del videojoc. Interessats en interactuar amb el mon.
- ***Socializers***: els hi agrada més interactuar amb altres persones que el propi rerefons del videojoc. Interessats a interactuar amb els jugadors.
- ***Killers***: els hi agrada competir contra altres jugadors. Interessats a actuar sobre els jugadors.

Com el nostre és un joc d'estratègia competitiu, veiem bastant clar a quin tipus de perfil ens hem de fixar. Una vegada fet això ens hem de preguntar com fer el millor joc possible per aquest tipus de jugadors.

És també important fixar-nos en l'edat d'aquest públic, ja que pot variar substancialment la manera com aquest reben els estímuls i reaccionen davant diverses situacions del videojoc. Per exemple, nens més petits poden acabar molt enfadats i acabar deixant el videojoc davant 3 partides seguides perdent o poden desinteressar-se si la idealització de les estratègies és massa complexa i se'ls hi escapa del seu abast.

Motivacions, necessitats i emocions del jugador

Hi ha diferents tipus de plaers que un videojoc pot atorgar. En el llibre "the art of game design" hi ha una llista de plaers bastant completa. Dels que apareixen podem veure que el nostre joc hauria de complir els següents:

- ***Delight in another misfortune***: és molt important en el joc competitiu. És un plaer que està basat en la desgràcia de l'altre. D'aquest rauen alguns altres plaers com el de la venjança davant la derrota.
- ***Possibility***: el plaer de tenir moltes opcions on triar. Base en els jocs d'estratègia.
- ***Pride in an accomplishment***: plaer o orgull en aconseguir realitzar una cosa. En els videojocs competitius freqüentment hi ha marcadors i rànquings, i la gent se sent molt orgullosa de la seva posició en ell.
- ***Thrill***. El plaer de l'emoció. En una partida igualada o complicada que es pot decidir tot en un últim moviment.
- ***Triumph over adversity***. Com l'anterior, però en una situació desigualada/injusta.

Per tant, hi ha algunes preguntes que poden servir de molt per començar a encarar el videojoc entorn el jugador. Això podria ser un resum de l'apartat anterior amb el més important a l'hora de dissenyar el videojoc.

- **In general, what do they like?** La possibilitat de tenir moltes opcions cap a la victòria, sentir-se superior al rival. El plaer de guanyar gràcies a les teves estratègies.

- **What don't they like, why?** Coses que tinguin a veure amb l'atzar. Qualsevol cosa relacionada amb atzar i percentatges que no estigui correctament equilibrat i justificat pot fer que el jugador se senti en un videojoc injust. Que per aconseguir tot o ser el millor s'hagi de pagar.

- **What do they expect to see in a game?** Un joc correctament equilibrat i amb moltes opcions per fer estratègies. Moltes mecàniques úniques i divertides.

- **If i were in their place, what would i want to see in a game?** Alguna cosa en el gènere que no hagi estat fet. Moltes possibilitats i rejugabilitat.

- **What would their like or dislike about my game in particular?** Introducció d'una mecànica nova en fer les estratègies. Cada personatge té habilitats úniques i sinergies. No s'ha de pagar per ser el millor ni per obtenir tots els personatges. A gustos, a algú els hi agradarà que siguin partides curtes i a altres no tant.

- **Why did your Player buy/play your game?** Mecàniques úniques, fresques i divertides pel gènere. Per molt entretinguda que sigui la saga *Worms*, fa molts anys que està estancada i no ofereixen nous continguts.

Hi ha un tipus de públic que ha aparegut amb la incorporació dels videojocs per a telèfons mòbils i que és segurament el públic que abraça més camp tot i que és una aposta poc segura. És el jugador **casual**; són persones que juguen en temps morts i per tant els hi agrada els videojocs de partides curtes i entretingudes, com és el cas que es treballa en aquest treball. El problema d'aquests jugadors és que es cansen ràpidament dels jocs i no solen invertir diners. Tot i això és molt important arribar a ells perquè fan que creixi molt la popularitat del videojoc i la seva comunitat.

2.5 REPÀS FINAL

Finalment per a fer un repàs final a manera de resum, es contesten unes preguntes que s'hauria de fer sempre en acabar d'idear un videojoc, són preguntes bàsiques però que defineixen els punts més importants per saber si podem entrar en el mercat amb el nostre nou producte. Aquestes les he recollit dels apunts del curs i les considero essencials per determinar si el nostre joc és una bona aposta.

- **Is it unique? Has anybody done it before?** El nostre joc és un pas següent a una saga de videojocs molt famosa i aclamada com es *Worms*. Han aparegut nous successors d'aquest però que no canviaven en essència. Amb el nostre joc volem incorporar mecàniques mai vistes i dotar-lo d'un nou nivell de profunditat.
- **Is it really fun and will it remain fun for the full game experience?** Incortpora les mecàniques divertides que van consolidar la saga *Worms* i alhora incorpora algunes noves mai vistes en el gènere.
- **Does it tap into something universally felt by your target audience?** Compleix amb tot el que busca el nostre públic objectiu. Estratègia i competitivitat.
- **Is the target audience large enough to make this a worthwhile project?** A part del públic específic fan del gènere, en ser un joc de mòbil i de partides curtes es busca enganxar al públic casual de mòbil que no té un gust definit del tot.
- **Can you pull it off technically?** Sí **Legally?** Sí **Financially?** I sí.

3. PLANIFICACIÓ

Aquesta etapa defineix l'estratègia seguida per arribar als objectius plantejats. Es descriurà breument el pla de treball, les tasques planificades, el temps estimat i els resultats esperats de cada tasca.

En un equip és on se sol posar en concordança tots els membres de l'equip per treballar de forma eficient, es reparteix el treball segons les disciplines, es fixen els terminis d'entrega, es consensua cada quant es faran reunions per veure el progrés, etc. En aquest cas com Battleland es realitza de forma individual no hi hauria reunions per exemple, però això no vol dir que no s'hagi de fer una planificació per igual, has d'organitzar un equip sencer, etc.

En un videojoc, trobem que com a qualsevol projecte hi ha les 3 etapes fonamentals, l'etapa de disseny, l'etapa d'elaboració, i l'etapa de retoc i proves abans de la seva publicació. La diferència radica en què en un videojoc veiem que hi ha moltes parts en el desenvolupament que s'executen de forma iterativa, que es vol dir amb això; que pot ser el cas que després d'implementar quelcom, tornis a la fase de disseny un altre cop. Per tant es podria dir que en els videojocs hi ha fases de proves i errors i continus canvis mentre es desenvolupa el videojoc, tot i abans d'arribar a la vertadera fase de proves i errors.

Per aquest motiu també és molt important la figura del dissenyador, perquè és qui ha de definir molt exactament els objectius de l'equip així com totes les coses que són primordials en el videojoc i ja més tard i si es pot permetre complementar-lo amb coses secundàries. Un bon disseny al principi fa que aquesta part sigui més ordenada, si no pot arribar a ser un caos.

Així doncs, veiem que amb una bona planificació aconseguirem ordenar correctament les prioritats i organitzar-nos col·lectivament i d'aquesta manera millorar en eficiència.

3.1 METODOLOGIA

Com hi ha un límit de temps molt ajustat, es decideix anar part a part sense tornar enrere per no estancar-nos en cap moment. Per tant, hem d'estar molt segurs en acabar una tasca per continuar a la següent, perquè no volem perdre temps en perfeccionar-la fins que no sigui al final del desenvolupament. Per a això hem de definir molt bé els requisits des d'un bon principi i les funcionalitats que hi haurà, una vegada complet amb això és on podríem en tot cas millorar el nostre videojoc afegint noves opcions.

La metodologia que s'explica en aquest mòdul és la que se sol utilitzar en els videojocs, i és similar a molts projectes de software en general, amb la diferència de què hi ha més quantitat de part artística-creativa. El desenvolupament també pot veure's afectat i variar depenent de la plataforma, el gènere, i fins i tot l'estil o perspectiva que s'utilitza. El procés mencionat se sol dividir en 6 fases: Concepció, disseny, planificació, producció i proves, publicació i manteniment.

1. **Concepció**: moment en el qual decidim embarcar-nos en el desenvolupament del videojoc. Motivat per les mateixes ganes de desenvolupar videojocs i pel treball de final de grau.
    a. Definim el que seria el concepte a grans trets. Pot ser una idea que hagi sorgit a partir d'experiències, pensar-la a partir d'una pluja d'idees, etc. En el meu cas contínuament em van apareixent idees en jugar a videojocs i m'apunto les que em semblen millors. D'entre altres opcions em vaig decantar per aquesta per al treball perquè no només es elaborar un videojoc, sinó enfrontar-se al problema de multiplataforma.
    b. Es consulta amb persones properes i enteses per rebre una mica de *feedback* [6]i es modela una mica més la idea fins a obtenir un concepte bastant més definit. En cap moment queda definit el videojoc, és tan sols una idea una mica ambigua amb moltes opcions a variar al llarg de la producció.
    c. Aquí entraria llavors l'estudi de viabilitat per atrevir-se a continuar endavant amb el projecte o buscar alternatives, ja que encara no s'ha fet

[6] El *feedback* en aquest context es la resposta que rebem dels jugadors, i a partir de la qual els dissenyadors han de saber respondre.

cap inversió i és un moment en què cal pensar detingudament i fredament.

2. **Disseny**: es detalla els elements fonamentals que acabaran component l'essència del videojoc, entre d'altres, els més importants són:
    a. Història
    b. Guió
    c. Art
    d. So
    e. Mecàniques
    f. Tecnologia

En aquesta fase és necessària la generació del document anomenat GDD (Game Design document). Aquest serà inclòs en el treball en forma d'apartat més endavant.

3. **Planificació**: Com he comentat, aquesta fase serveix per a definir la metodologia de treball i dur a terme un disseny i desenvolupament de videojoc el més òptim possible.
    a. Definició i assignació de tasques.
    b. Definició dels temps estimats per dur a terme les tasques. Dates claus, on s'ha d'entregar la feina sol·licitada. També és important definir els punts de no retorn, és a dir, els determinats moments en el desenvolupament que s'ha de puntualitzar que no hi haurà més canvis en certes etapes de producció.

    S'ha de tenir en compte que hi poden haver imprevistos, i també que algunes tasques poden requerir unes altres i alguna persona potser no pot començar la seva part sense la d'un altre. Aquestes dependències s'han de tractar correctament amb una bona planificació i permetre el treball paral·lel dels diversos membres de l'equip. Hi ha eines de diagramació d'activitats per a poder veure la planificació de forma visual i de forma entenedora, en aquest treball utilitzarem el diagrama de *Gantt*[7].

[7] El diagrama de *Gantt* és una eina grafica per exposar la planificació.

4. **Producció i proves:**
    a. És l'hora de treballar... implementació de les tasques definides amb anterioritat. Programació, il·lustració, interfície, art i animacions i so solen ser les parts essencials i les quals solen ser dividides per grups o persones que s'encarreguen específicament.
    b. Hi sol haver control de producció, petites reunions per veure que es porta al dia tota la feina.
    c. En la fase de producció hi ha també una etapa de *testing*, que solen fer els propis membres de l'equip, o si hi ha, els testers.
5. **Publicació**:
    a. Etapa de màrqueting: part fonamental per donar a conèixer el videojoc i aconseguir el màxim nombre possible de jugadors. Molts cops hi ha un departament propi encarregat d'aquesta feina.
    b. Optimitzar el videojoc per a les plataformes concretes
    c. Distribuir
6. **Manteniment (Post llançament):**
    a. Anàlisis del *feedback*, i amb aquesta retroalimentació definir els passos següents a seguir.
    b. Manteniment del joc, corregint errors, etc. Amb la publicació apareixen molts errors i desnivells en el balanceig del joc que poden no haver sigut contemplats anteriorment.
    c. Actualitzacions perquè el videojoc sigui tenint novetats i el públic no es cansi.

Segons les fonts d'informació, aquest procés pot variar en el nombre de fases, però al final sempre és el mateix procediment, únicament apareixen diferents classificacions o agrupacions d'apartats i per això en laguns llocs poden aparèixer alguna fase més.

I òbviament, tot i que durant tot l'apartat s'ha parlat en el terme general d'un equip de desenvolupament, el videojoc està realitzat per una persona així que hi ha coses que s'obviaran, per exemple no hi ha treball paral·lel entre programadors, reunions entre membres de l'equip, etc.

3.2 PLA DE TREBALL

En aquest apartat hem de planejar les coses que farem, així com el temps que ens portarà dur-les a terme i quin serà el mètode a seguir per cada cosa. Hem de considerar tots els escenaris possibles, ja que en un desenvolupament de tal calibre poden sorgir molts problemes inesperats, així com acurtar temps en tasques que en un principi es podria pensar que portarien molt més temps (per exemple, la implementació d'una mecànica molt complicada però que en consultar internet en cerca de fonts d'informació trobar llibreries o eines amb la feina ja feta).

Els videojocs solen tenir un procés iteratiu per cada una de les tasques (Figura 11). Per exemple, es dissenya un personatge, es genera un prototip i s'avalua, i llavors potser tornem a la fase de disseny perquè hi ha alguna cosa que no queda com pensàvem, i així fins que s'obté un resultat satisfactori.

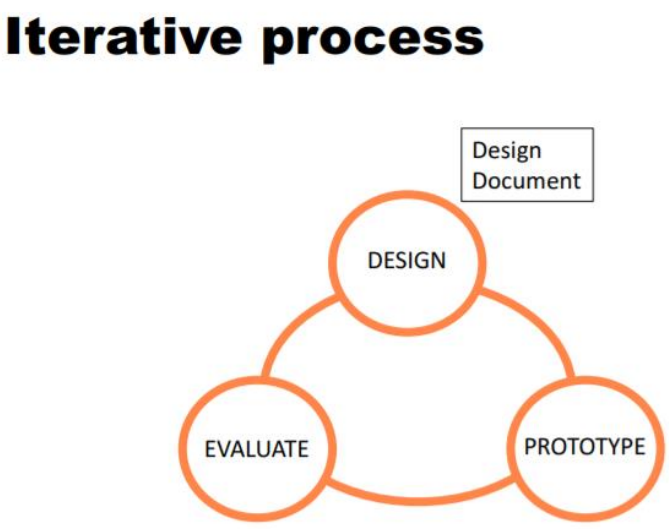


Figura 11 Procés iteratiu

3.3 ABAST DEL PROJECTE

Per fer una bona planificació, primer s'ha de tenir molt clar l'abast del projecte. S'ha de tenir ben definit totes les coses que apareixeran en el videojoc, ja que així podem organitzar-nos correctament per a fer una correcta anàlisi i fer una correcta extracció de requisits del projecte. Una vegada estiguin definits els punts més importants, diferenciar i classificar les diferents tasques a realitzar es torna molt més senzill. És crucial no fer canvis als elements conceptuals del nostre joc una vegada s'ha començat el procés d'elaboració del disseny.

A continuació s'exposa un recital de punts importants que ha de contenir el videojoc, i que es la seva definició en essència.

3.3.1 DOCUMENT DE CONCEPTE

Videojoc 2D d'estratègia militar per torns, concretament inspirat en W*orms*. Té com a objectiu entretenir al jugador i proposar-li reptes de tipus estratègic. Està enfocat purament al multijugador competitiu, per lo que, ha de tenir el factor de rejugabilitat molt cuidat, ja que com a joc no té un final definit. Té dos modes de joc, multijugador local *hotseat*[8] i multijugador online, ambdós competitius.

El joc es basa a guanyar partides en el multijugador Online per intentar arribar a ser el millor. Tindrem un perfil de jugador i hi haurà un rànquing o una manera de determinar qui és el millor. Depenent del resultat de la partida pujarem o baixarem en aquesta posició. També es podrà guanyar monedes que les podrem bescanviar per personatges en la tenda.

Les partides seran de 3 personatges contra 3 personatges. Cada personatge tindrà unes característiques úniques, així com una habilitat especial i una habilitat passiva. No es podrà repetir personatge en una mateixa formació, tot i que és possible enfrontar-nos al mateix personatge però en l'equip contrari. Es busca crear partides curtes i experiència divertida i desafiant.

El joc estarà disponible per ser jugat des de ordinador o des de dispositiu tàctil amb Android, permetent partides creuades entre plataformes. També es podrà jugar amb un

[8] Es diu multijugador *hotseat* a aquell que la partida es desenvolupa en un sol dispositiu.

sol dispositiu. No es pot jugar contra la màquina. Tots els controls es basaran en botons virtuals.

Hi haurà molts personatges molt diferents així com diferents escenaris. Les diferents versions del joc no diferiran en contingut. Tot l'art serà estil *cartoon*, i és un joc que pot jugar qualsevol persona sigui quina sigui la seva edat. La càmera serà en 2D frontal. Sempre hi haurà música simple i ambiental, per no distreure massa al jugador. Hi haurà diferents sons per indicar diferents situacions (s'acaba el temps del torn, t'han ferit, etc.).

Com tenim un perfil de jugador i hi ha rànquings i tal, tota la informació es guardarà en servidors al núvol de forma automàtica en acabar partides o fer transaccions. El joc es podrà jugar sense internet, però només l'opció de combat en mode local.

Interfície molt simple, no carregada d'informació. Només dades especialment importants per a l'usuari, com a qui li toca actuar i quin temps li queda. Els menús, al igual que com ja s'ha comentat, funcionaran tots amb botons virtuals. En el transcurs de la partida també, apareixeran objectes que oferiran potenciacions als personatges.

3.4 TASQUES PLANIFICADES

Ara que tenim els punts més importants que abraça el nostre joc, és molt més fàcil dividir-lo per parts i organitzar les tasques.

Per a projectes de tanta grandària i en el qual hi ha tasques molt diferents les quals solen ser repartides entre departaments, és convenient treballar amb el que s'anomena *milestones*; la seva definició bàsicament és que són fites a curt termini. Així, podem establir períodes curts de temps per anar implementant a poc a poc el videojoc i de forma organitzada, establint alhora prioritats. D'aquesta forma podem passar d'objectius complexos a petites implementacions senzilles i rapides de completar i alhora així aconseguim veure millor el progrés.

Per a aquestes tasques també hi ha eines o serveis a la Internet que ajuden a organitzar-te tant si treballes individual com col·lectivament. En el meu cas m'he recolzat en el servei web ***Trello***, per anar establint-me objectius i períodes per dur-los a terme. A continuació s'adjunta una captura per veure el funcionament del servei (Figura 12)

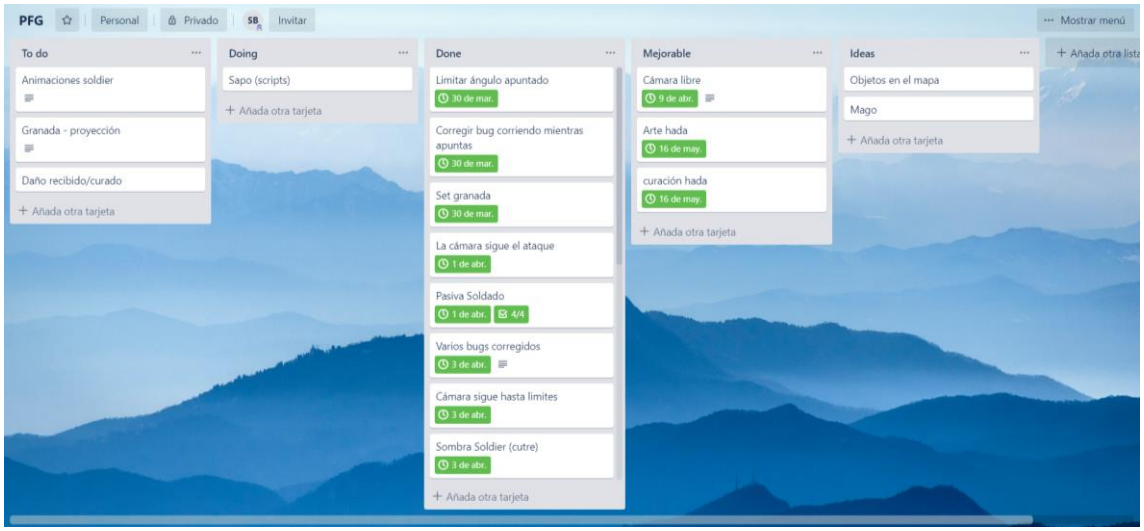


Figura 12 Captura de la utilització de Trello en un moment determinat del procés

És molt important no anar deixant errors o bugs [9]a mesura que es va implementant i anar a la següent tasca sense fer les corresponents proves, ja que això pot provocar l'efecte bola de neu, passarà factura més endavant i potser entra en conflicte amb altres coses i no deixa a un company fer la seva tasca. També, i sobretot quan es treballa de forma individual, és molt important evitar el multitasking (fer moltes coses alhora), és important saber que toca fer i no deixar-se portar per l'emoció; el que vull dir és que a tot desenvolupador li agradaria programar el seu joc veient ja el seu personatge animat, però hem de saber que és més important al principi, i per això hem de saber que no és eficient treballar en la mecànica del personatge a la vegada que estem fent les animacions. És important anar fent cada cosa en el seu moment corresponent.

Així doncs a continuació es detallaran les tasques planificades i les subdivisions en les quals s'ha organitzat per a agilitar el procés de treball. Per fer-ho s'ha creat una taula (Taula 5) amb valors orientatius.

[9] *Bug* és el terme més utilitzat per fer referencia a errors de desenvolupament en un videojoc.

Número	Tasca	Temps estimat
01	Preparació de l'entorn d'Unity. Crear una petita escena de proves, definir la càmera, etc.	6 hores
02	Desenvolupament en primera instància de personatge general. Objecte que es mogui com a màxim una certa distància, crear-li les físiques i *colliders*. Implementació d'atac bàsic.	8 hores
03	Creació d'interfície simple. Botons per moure el personatge i per fer atac bàsic. Indicadors de temps i torn, així com requadre amb informació del personatge i barra de vida. Fletxa indicadora de qui li toca actuar.	10 hores
04	Implementació d'un controlador de joc per al mode local. Estableix a quin jugador li toca actuar, controla el temps de cada torn, el torn en què s'està, etc.	20 hores
05	Desenvolupament personatge #01. Assignació de paràmetres personalitzats. Creació de l'habilitat especial i habilitat bàsica. Creació de l'art i animacions. Així com a efectes i elements únics del personatge. Controlador d'habilitats en la interfície.	30 hores
06	Desenvolupament personatge #XX. Per cada personatge s'ha de fer més o menys el mateix procediment de coses, però un personatge o un altre pot variar bastant el temps de dedicació, ja sigui per culpa de les animacions i a l'hora d'implementar les seves mecàniques pròpies. Per tant es posa unes hores en forma de mitjana esperada.	30 hores * personatge
07	Modificacions necessàries i configuració del joc per poder executar-se des de dues plataformes.	8 hores

08	Creació de la pantalla de formació i estratègia. Es mostren les característiques de cada personatge. Permet formar diferents equips agrupant 3 personatges.	20 hores
09	Creació de pantalla principal quan s'inicia el joc. Art i implementació.	10 hores
10	Creació de pantalla del menú principal. Art i implementació.	10 hores
11	Creació de pantalla d'opcions. Art i implementació.	20 hores
12	Creació de controlador de joc online. Modificacions corresponents i implementació del joc online i creuat.	40 hores
13	Creació de perfil de jugador. Creació i interconnexió del videojoc amb bases de dades al núvol. Implementació de sistema de rànquing i tenda de monedes.	40 hores
14	Interfície definitiva del camp de batalla i dels menús.	16 hores
15	Inserció de música ambiental, efectes sonors i altres elements.	10 hores
##	TOTAL	248 hores

Taula 5 Tasques planificades

S'ha de tenir en compte que aquí només hi ha les hores pel que fa al desenvolupament, no hi ha de les etapes de màrqueting, ni proves, etc.

És important remarcar, que no són les hores invertides en el prototip, són les hores que s'esperarien necessàries en un principi per implementar els components més essencials del joc. Al moment de fer la planificació s'ha comprés que el projecte se surt una mica de les possibilitats de poder tenir una versió publicable en temps del projecte i que compleixi tots els requisits, pel qual a l'hora de fer el prototip s'haurà de prioritzar continguts.

Per tant, si agafem els números que ens sortien a les inversions inicials, i multipliquem el nombre d'hores per el sou/hora, trobem que en l'hipotètic cas mencionat, haurie de pagar al personal uns 13.000 €

3.5 DIAGRAMA DE *GANTT*

En aquest subcapítol, s'exposarà diferents diagrames de *Gantt*, els quals són produïts mitjançant el servei web online.officetimeline.com de forma gratuïta.

3.5.1 PROJECTE

En aquest diagrama (Figura 13) s'abraça absolutament tot el projecte, des de la part de concepció, fins a la part de proves. Està també incorporat la feina d'haver de realitzar aquesta memòria, que bàsicament és durant tot el que dura el projecte.

Com el treball es realitza en el temps lliure de l'alumne, s'ha decidit dedicar dues hores diàries contant caps de setmana per a la realització de cada tasca, que se sumen al temps que s'inverteix al fer una mica d'aquesta memoria cada dia.

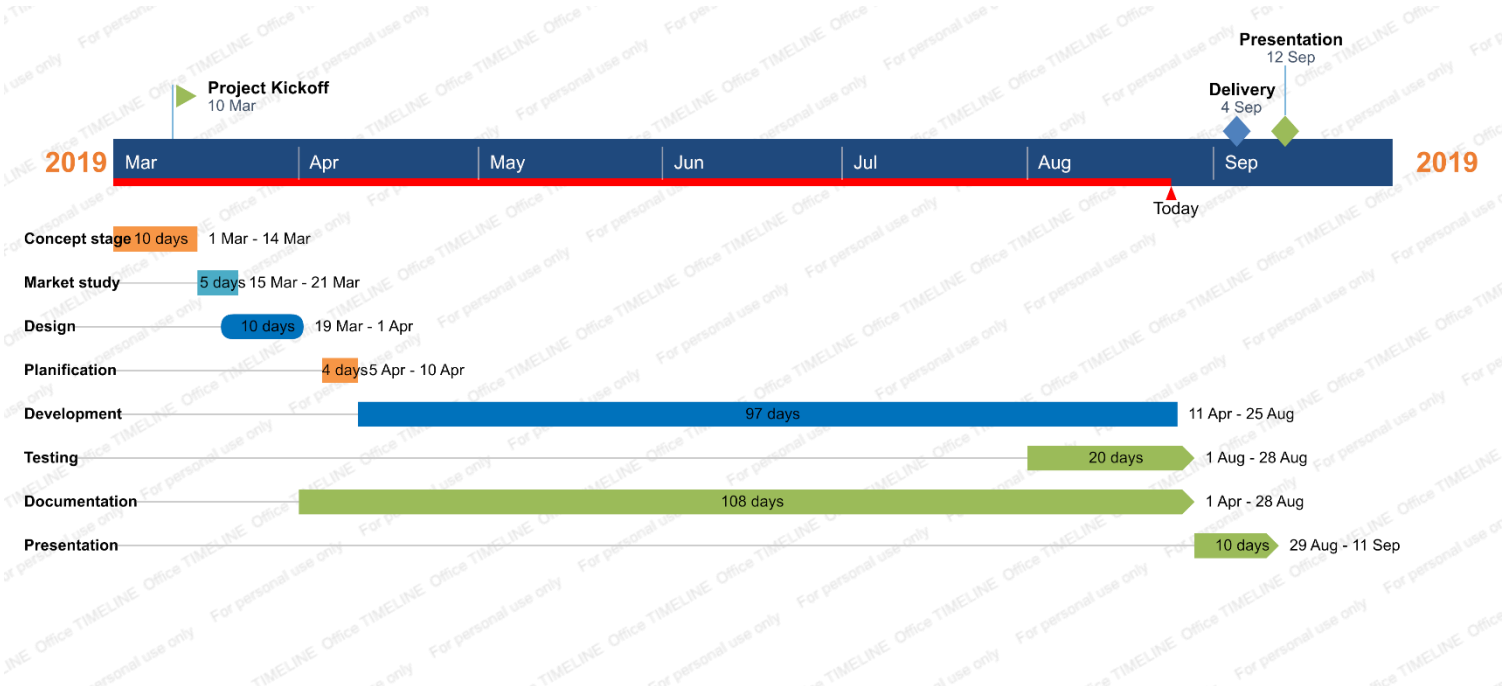


Figura 13 Diagrama de Gantt del projecte

3.5.2 DESENVOLUPAMENT

En el seguent diagrama s'indica el temps esperat que es necessitaria per a produir les tasques que s'han definit previament en l'apartat de tasques planificades, en referent al desenvolupament del joc.

Com es pot veure, l'abast del joc és més gran que els limits temporals que s'imposen per a realitzar el projecte. Aixo no es cap inconvenient, perque el que s'ha calculat a la part de tasques planificades és el que faria falta per tenir la primera versió oficial del joc publicable, però en aquest treball no busquem aquest objectiu, sino fer el disseny del joc i llavors fer un prototip del *gameplay* per ensenyar la idea en execució. En el corresponent apartat quan es parli del prototip s'indicara quines tasques han pogut ser realitzades, aixi com a que se li ha donat més prioritat, etc.

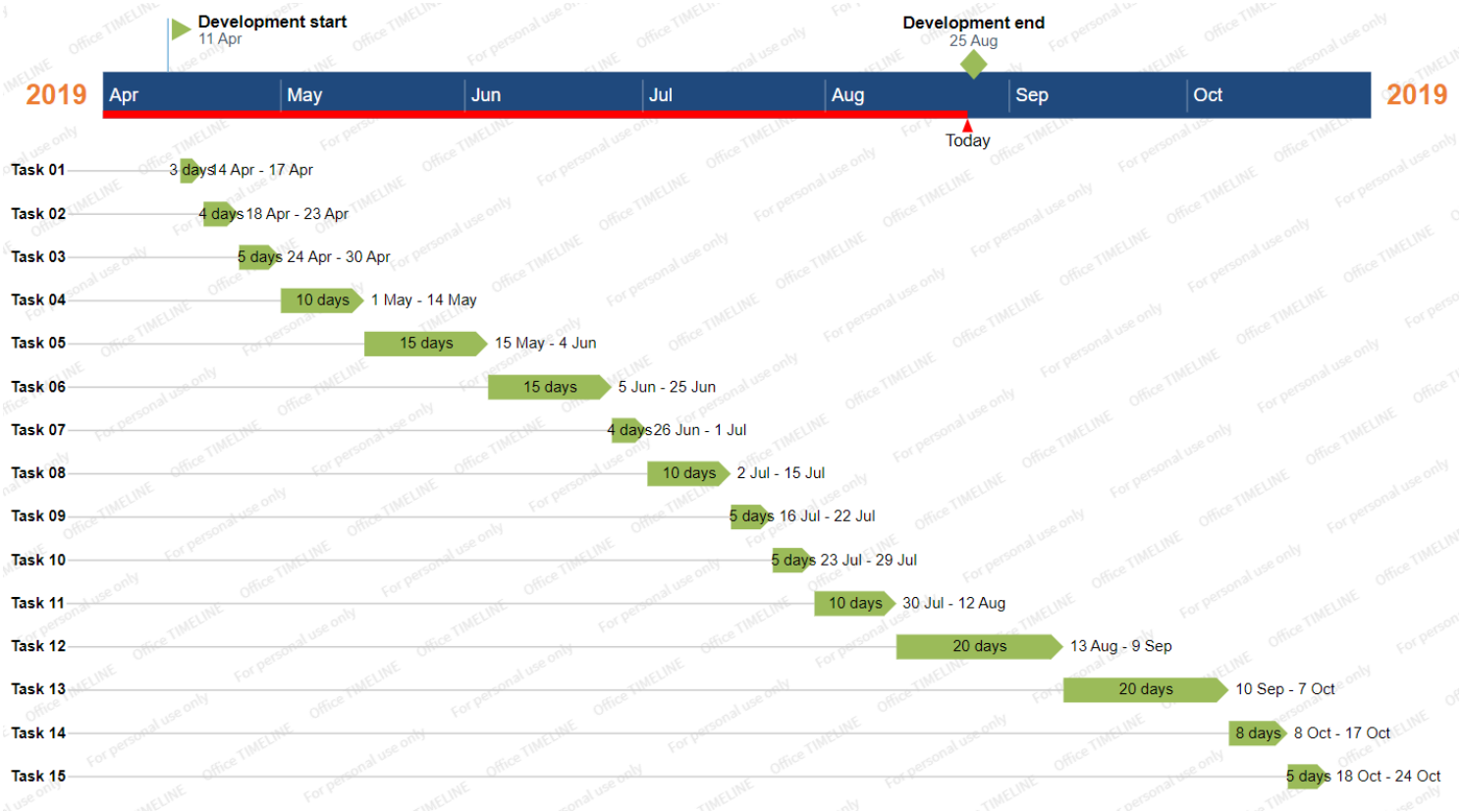


Figura 14 Diagrama de Gantt del desenvolupament

4. MARC DE TREBALL I CONCEPTES PREVIS

Aquest punt, que és l'anterior al que és parlar sobre el projecte en si mateix, servirà principalment per a preparar al lector i introduir els conceptes necessaris per a poder comprendre de forma correcta i fer un bon seguiment del document de disseny del videojoc.

D'aquesta manera, s'introduirà una mica el tipus de joc que es realitza. Això es farà exposant primer el màxim exponent del gènere i explicant el seu *gameplay*, després d'això s'exposarà quines diferències s'introdueixen en la meva proposta de videojoc respecte al *gameplay* comentat, a l'igual que les similituds.

També s'explicarà quin motor s'ha decidit utilitzar per a desenvolupar el videojoc, conjuntament amb les raons per haver fet aquesta elecció.

En aquest apartat i com en realitat ja s'ha exposat en aquest document en anteriors moments, s'especifica que el treball és realitzat en la seva plenitud per una sola persona i la màxima participació de terceres persones en el disseny són conversacions amb altres persones en les quals es recollien idees i opinions.

4.1 REFERÈNCIA

La idea per a un nou videojoc pot sorgir de moltes maneres diferents. Pot ser que tinguis una idea completament nova que revolucioni un gènere o pots imitar jocs consolidats i donar-los un cop de rosca. Pot haver la possibilitat que se t'ocorri un tipus de *gameplay* encara no definit i crear el teu propi subgènere en la indústria. El cas que es treballa en aquest document és el segon, és el d'agafar un videojoc el qual per si mateix ja ha funcionat molt bé i afegir-li novetats i millores.

D'aquesta manera veiem que ja tenim una base molt important per al nostre joc, ja que hi ha moltes parts que haurem de tan sols imitar o adaptar, i d'altres que seran les significatives i seran les que deixin la marca personal i distintiva respecte als altres videojocs semblants.

A continuació s'explica el funcionament d'una partida normal del videojoc *Worms*, videojoc que ja ha sigut mencionat amb anterioritat perquè és la referència principal de la qual va sorgir la meva idea de joc. Juntament amb idees personals que vaig veure que podien encaixar de manera molt fructífera, s'aconsegueix crear un joc molt més estratègic i que ofereix molta més personalització, i com a conseqüència partides més úniques i entretingudes.

Figura 15 Captura d'un joc de la saga Worms

És un joc en el qual hi ha dos equips sobre un camp de batalla. Aquests equips estan formats per tres personatges iguals, amb les mateixes característiques. Llavors el *gameplay* és el següent:

1. És el torn d'un personatge d'un equip, aquest pot moure's i atacar, després d'això o en acabar el temps que té per actuar, s'acaba el seu torn.
2. Llavors és el torn del personatge de l'equip 2. Cal remarcar que el jugador no pot decidir quin personatge vol moure, sinó que va amb un respectiu ordre. Perquè quedi més clar vindria a ser així:

Equip1:Personatge1->Equip2:personatge1-> Equip1:personatge2->Equip2:personatge2

I així successivament.

3. Els personatges per atacar poden seleccionar entre diferents armes, cadascuna amb les seves particularitats o limitacions.

4. Els personatges moren quan la seva vida arriba a 0 o bé surten dels límits del mapa (per exemple quan cauen a l'aigua)
5. La partida acaba quan un equip es queda sense personatges disponibles per lluitar.

Algunes particularitats del *gameplay* que el fan bastant únic és que els escenaris es veuen afectats per l'efecte de les explosions, modificant d'aquesta manera la seva forma.

El meu videojoc respecta la jugabilitat de manera precisa. Amb això vull dir que bàsicament el *gameplay* és el mateix, l'únic que canvia són els personatges. Els personatges tindran cadascun característiques úniques, si bé en el *Worms* tots els personatges tenen la mateixa vida i poden moure's el mateix número d'unitats, en Battleland cada personatge tindrà les seves pròpies dades. Però no només això, no només diferiran en aquests paràmetres bàsics, cada personatge tindrà les seves pròpies habilitats. Si bé en el W*orms* tots els personatges poden usar les mateixes armes, en Battleland cada personatge a part d'un atac bàsic tindrà una habilitat especial, això és l'element clau del joc que el converteix en un joc molt més tàctic. Hi haurà unitats que potser tindran menys vida però la seva habilitat permet curar els aliats, llavors el jugador ha d'identificar com col·locar aquests personatges sobre el camp de batalla i si adoptar postures defensives o ofensives depenent de la seva pròpia composició de personatges i la de l'enemic.

Continuant amb les diferències, en el meu no es podrà destruir ni alterar l'escenari, ja que el jugador mateix pot utilitzar aquest en la seva estratègia, per exemple per amagar alguna unitat feble darrere d'una paret. En Battleland els personatges tampoc podran saltar, per tant la col·locació dels personatges és essencial.

I finalment, *Worms* és una saga molt aclamada i molt important al llarg de la història dels videojocs, ha aparegut en moltes plataformes i ha tingut moltes adaptacions, però encara mai ha ofert la possibilitat de jugar amb jugadors d'una altra plataforma en una mateixa partida. Battleland i com a part d'investigació d'aquest treball, oferirà la possibilitat de creuar partides entre jugadors de dispositius mòbils i jugadors d'ordinador.

4.2 MOTOR DE JOC: UNITY

Figura 16 Logotip d'Unity

El videojoc s'ha decidit implementar amb el motor de videojocs *Unity*, motor desenvolupat per *Unity Tecnhonolgies*. A continuació s'explica perquè he pres la decisió d'emprar *Unity* i també algunes coses essencials a conèixer sobre el motor.

En l'actualitat, la majoria de videojocs arreu del món són desenvolupats o bé per *Unity* o bé per *Unreal*, amb excepció de les grans empreses que es poden permetre tenir un departament sencer per a desenvolupar un motor de joc propi. Segons la pròpia pàgina web *d'Unity*, esmenten que més de la meitat de videojocs del món són desenvolupats amb la seva eina.

He decidit emprar *Unity* perquè és una tecnologia que ja coneixia amb anterioritat perquè l'hem utilitzat durant el grau i amb la qual em sento molt còmode treballant. A continuació s'explica algunes característiques per el qual el motor ha esdevingut un dels millors i justifiquen l'elecció.

- Permet desenvolupament multiplataforma d'entre gairebé totes les opcions que hi ha en el mercat, tant de consoles com dispositius mòbils. A més, exportar el mateix videojoc a diferents plataformes és molt fàcil de realitzar i ho fa a molta velocitat, deixant gairebé tota la feina a l'adaptació de controls i interfícies.
- Permet desenvolupar tota mena de videojoc, té diferents mòduls per treballar tant en 3D com en 2D. Integra exemples de molts tipus de gèneres i implementacions bàsiques per començar amb el teu desenvolupament.
- Té una versió gratuïta que és suficientment capaç per si mateixa. Pots fer qualsevol videojoc sense cap restricció, però apareixen limitacions com que apareix el logo *d'Unity* en començar el joc, o els serveis que ofereixen estan limitats (per exemple només et permet tenir servidors dedicats de màxim 20 persones a la vegada).

- L'editor (el que es presenta en la figura 17) és molt intuïtiu i senzill d'utilitzar. Permet treballar amb molt tipus de fitxers i per tant es pot complementar el motor amb molts programes externs com *Photoshop*, *Blender*, etc.

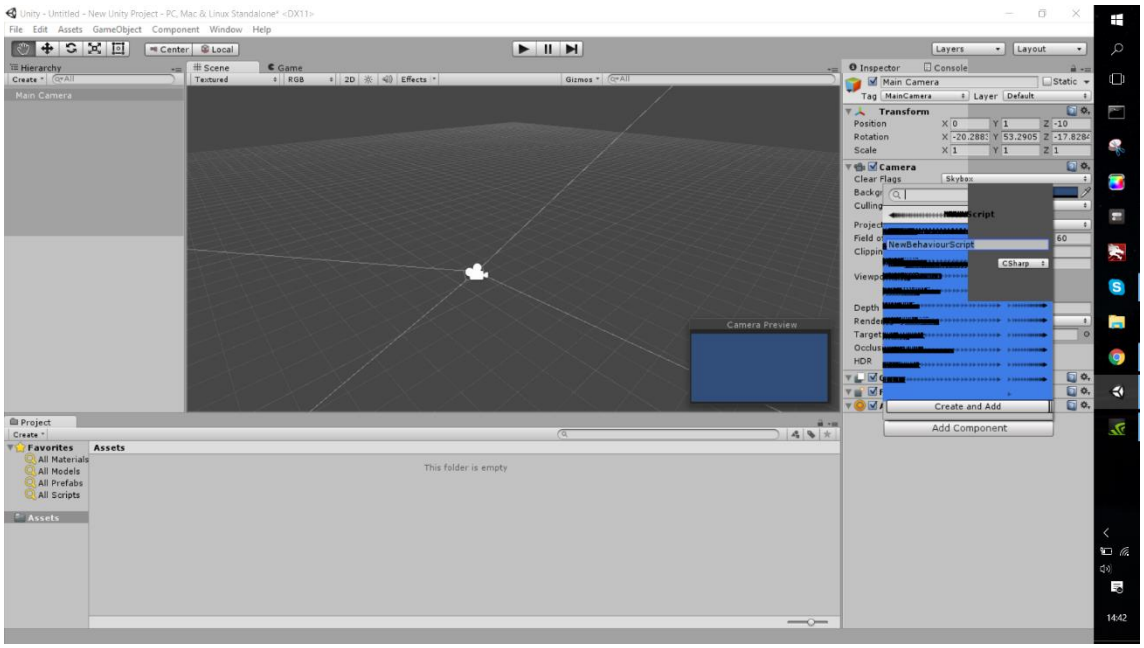


Figura 17 Captura de l'editor d'Unity

- Té una gran documentació oficial, però possiblement i el més important, té una comunitat enorme de desenvolupadors i hi ha molta informació a la Internet i pràcticament pots resoldre tots els teus dubtes cercant a la xarxa.
- El llenguatge de programació que empra és c#, amb el qual ja tinc bastant interioritzat.

Per treballar amb *Unity* s'ha de conèixer uns conceptes fonamentals [4]:

1. *Gameobjects*: cada objecte que és a l'escena. Això comporta tant personatges, com llums, com l'escenari, etc. Els *gameobjects* per si mateixos no fan res, aquests estan formats per ***components***
2. *Components*: aquí estan implementades les funcionalitats. Defineixen el comportament dels *gameObjects*.
3. *Variables*: els components tenen diferents paràmetres que es poden modificar tant des de l'editor com a través d'*scripts*.

4.3 PLATAFORMES

En l'apartat d'introducció, es dedica un petit capítol a explicar perquè el videojoc és multiplataforma, en aquest moment però, el que es detallarà a continuació són les multiplataformes a les quals va dirigides el videojoc i els motius per a la seva elecció.

Battleland es pot jugar des de:

- Dispositiu amb Sistema Operatiu Windows 64 bits
- Dispositiu amb Sistema Operatiu Android

El videojoc es pot jugar amb aquestes diferents plataformes amb la idea de resoldre un dels objectius plantejats en un inici, com és el d'unir comunitats de plataformes diferents.

Per què aquestes plataformes?

- La interacció es fa mitjançant els botons virtuals que es dibuixen a la pantalla del joc. Ideal per a aquestes dues plataformes, perquè es pot jugar amb inputs tàctils o un ratolí que simuli aquests mateixos.
- Són les plataformes més utilitzades a la majoria del món.
- Unity permet una fàcil exportació a aquestes plataformes sense haver de pagar llicencies o quelcom semblant.
- Es disposa d'ambos dispositius per a poder realitzar les proves pertinents.
- No s'ha d'adaptar específicament. És el mateix joc realment, tan sols compilat en un format i arxiu diferent.

El videojoc està pensat principalment per als dispositius mòbils, es podria dir que la versió de Windows, és la versió d'Android "virtualment" jugada des d'un ordinador.

Realment Unity ofereix exportar a molt tipus de plataformes diferents amb molta facilitat, incloent-hi videoconsoles d'última generació. Per a aquest tipus de maquinari, sí que caldria realitzar diferents canvis en l'àmbit de disseny, ja que en no poder emular els inputs físics caldria redissenyar la interacció, per exemple fent que el joystick [10]dels controladors mogués el personatge en comptes de polsar el botó virtual de la pantalla.

[10] El *Joystick* és un periferic d'entrada que consisteix en una palanca que gira sobre una base i informa del seu angle i dirección al dispositiu que la controla

4.4 PEGI

Figura 18 Logotip de PEGI

Per a fer una correcta regulació dels videojocs, es va crear el sistema PEGI (Pan European Game Information). És un sistema d'autoregulació, és a dir, la pròpia desenvolupadora el defineix 53erquè el consumidor pugui saber quin tipus de contingut apareix i per a quina edat està recomanat, després però passa per un procés de comprovació que entrega a l'editor la llicència que autoritza a utilitzar les etiquetes PEGI.

Per a tal comesa, es disposa de dos tipus d'icones descriptors:

- Edat recomanada
- Contingut del joc

Així, abans de comprar un videojoc el consumidor pot veure si el videojoc és apte per a ell mateix o per a qui va dirigit veient les etiquetes (Figura 19), i així pot utilitzar aquesta informació per a decidir si jugar-lo o no. En cap moment aquest indicador és prohibitiu, és a dir, encara que posi que el joc està recomanat perquè el juguin persones majors de 18 anys, una persona de 16 pot anar a una tenda i comprar-lo sense cap problema.

Figura 19 Etiquetes PEGI

En el cas de Battleland, per l'estil del videojoc i pels elements que apareixen en el seu interior, hauríem de senyalar que és **PEGI 3** i també que conté **joc en línia**.

5. DISSENY DEL VIDEOJOC

A continuació es presenta el capítol que en realitat és el document de disseny del videojoc.

S'ha omès les parts que han sigut comentades en un altra secció, com per exemple el públic objectiu i perfil del jugador, o com l'estudi de mercat.

5.1 ESTIL

S'ha decidit començar per l'estil per donar a entendre ràpidament el concepte general del joc i el lector pugui començar a imaginar-se'l en el seu cap.

En aquest subapartat s'explica els diferents aspectes que en conjunt definirien l'estil del videojoc. Trobarem comentats el gènere, l'estil visual i la perspectiva i posició de la càmera.

L'estil moltes vegades és igual o més important que el propi gameplay, perquè és la imatge del joc, però és també qui determina el funcionament d'aquest en molts casos, i defineix moltes funcionalitats, per posar un exemple, si l'estil està definit com per a jugadors petits, podem simplificar molt les físiques dels objectes.

5.1.1 GÈNERE

Battleland pertany al gènere d'estratègia i dins d'aquest, al subgènere anomenat estratègia militar per torns. Dintre d'aquest gènere hi ha molts tipus de videojocs que poden diferir bastant en les mecàniques, així que com s'ha dit durant tot el treball, es podria especificar encara més i dir sense gens de por que el videojoc està clarament inspirat en la saga *Worms*, introduint nous conceptes i funcionalitats.

En Battleland així com en *Worms*, dos jugadors s'enfronten controlant varis personatges durant cert temps i comparteixen l'objectiu d'eliminar els personatges de l'equip rival.

Es fusiona el concepte de *Worms* amb un altre concepte que ha esdevingut especialment popular en els últims temps, que és la idea de personatges amb particularitats úniques. Si bé això ja ha existit des de bastant temps, ara mateix s'ha convertit en tendència i molts dels jocs més populars del món utilitzen aquesta característica. La figura 20 mostren com dos dels jocs més jugats en l'actualitat tenen un gran número de personatges diferents.

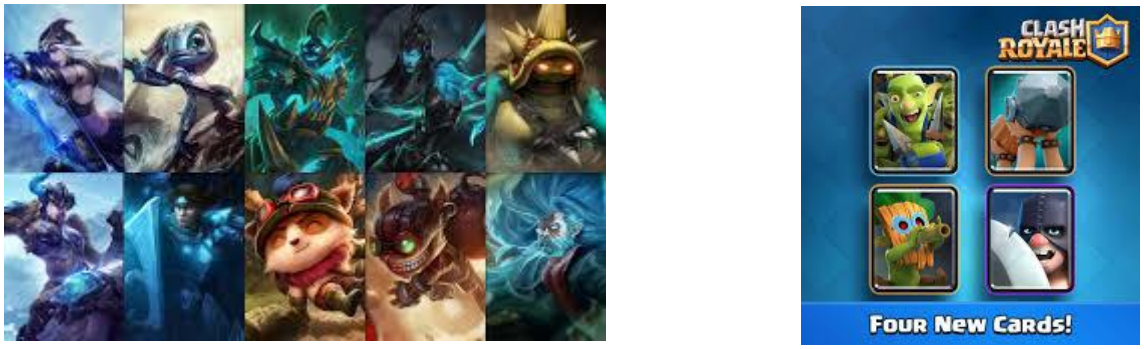


Figura 20 Captura amb diferents personatges dels jocs League of Legends i Clash Royale

5.1.2 CÀMERA

La càmera que s'utilitza en el videojoc és la possiblement més utilitzada al llarg de la historia, la perspectiva 2D i frontal que es veu en la figura 21, encara que juga amb avantatge perquè altres que empren el 3D han estat limitades a la potència tecnològica del moment. Aquesta perspectiva ha sigut sobretot molt utilitzada per videojocs de plataformes.

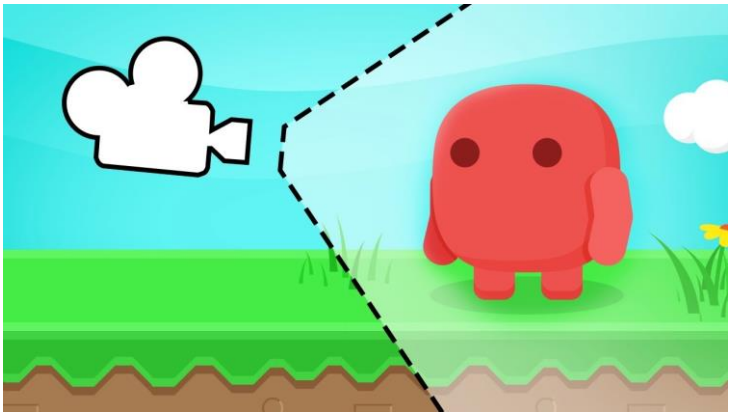


Figura 21 Representació de la càmera 2D frontal

Com he dit, dins el gènere d'estratègia militar per torns poden aparèixer molts tipus diferents d'estil. Battleland sorgeix amb la idea de donar un cop de rosca a la fórmula creada a la saga *Worms*, per tant imita el seu format.

Tot i que hi ha molts tipus d'estils dins el gènere, després de fer una cerca utilitzant les paraules clau: *videojuegos estratègia militar por turnos*, he pogut arribar a la conclusió de que la càmera més emprada sol ser amb vista isomètrica pel número de videojocs que he vist que la utilitzen, penso que és perquè com solen haver moltes unitats tant aliades com enemigues, aquest tipus de càmera ofereix el fet de poder observar amb detall el mapa, i sobretot poder veure amb precisió la posició de les teves unitats i les de l'adversari i calcular molt bé els següents moviments. A la figura numero 22 podem veure un exemple d'aquest tipus de camara en un joc d'estrategia militar per torns.

Figura 22 Càmera isomètrica

Com he dit, al tenir com a estil de referència el joc W*orms*, la càmera imitarà el seu funcionament. A la figura 23 veiem una comparativa per veure com s'imita l'estil de la càmera.

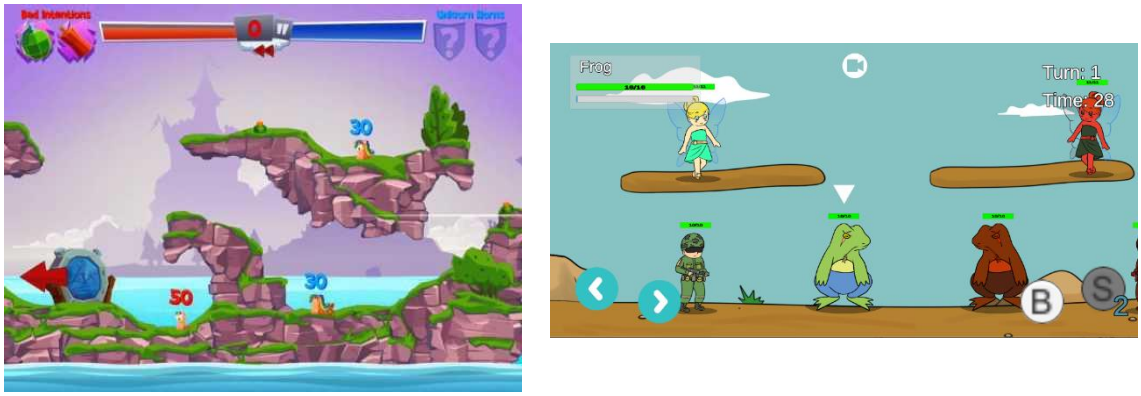


Figura 23 A l'esquerra, captura d'un joc de la saga Worms, a la dreta, captura del joc Battleland

Per a assolir tal objectiu pel que fa a la representació, en el motor de joc haurem d'indicar que la càmera ha de seguir una projecció ortogràfica en lloc de amb perspectiva, ja que els personatges només tenen la seva instància en el mateix pla, i en tot cas si volem indicar certa profunditat, només es veurà reflectida en l'escenari o paisatge, i aquesta sensació de perspectiva es farà en la mateixa imatge que representi en aquest cas el paisatge.

5.1.3 GRÀFICS

Amb gràfics es vol fer referència al tipus d'estil artístic del qual disposarà el videojoc, a la figura 24 en podem veure uns quants d'exemple.

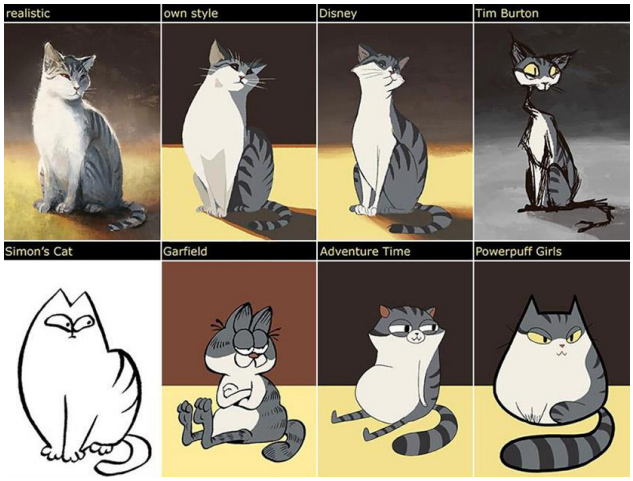


Figura 24 Comparativa d'estils

Igual que ha passat amb els diferents subapartats del capítol actual, i en realitat com sol ocórrer sempre sigui quin sigui el videojoc, com hem vist un gènere no determina l'estil, ni molt menys llavors els gràfics que aquest tindrà.

En Battleland es busca un estil gràfic poc realista, concretament de dibuixos animats, com el que es veu a la figura 25. Considero que com és un videojoc pensat perquè pugui jugar gent de moltes diferents franges d'edats és l'estil convenient. A sobre, encaixa amb la idea de ser un videojoc amb humor i personatges carismàtics.

Figura 25 Estil de Battleland

Tot i que seria molt bo comptar amb uns gràfics molt cuidats i detallats, els gràfics són realitzats per mi i per tant no es pot dedicar el temps que a un li agradaria i per això i

com s'ha detallat al quadre d'autovaloració els gràfics seran bastant simples. Tot i això, aquest fet no desentona amb el resultat final, ja que en ser un videojoc pensat per a ser jugat en diferents plataformes, i d'entre d'elles els dispositius mòbils, no es vol carregar massa la part gràfica del videojoc, ja que ha de poder ser executat en bones condicions en qualsevol dispositiu que compleixi unes especificacions mínimes. També, com ha de poder veure's en pantalles petites (de 5 a 6 polzades normalment en els dispositius mòbils), pot ser bastant innecessari dedicar molt de temps en petits detalls que poden passar inadvertits.

No obstant les limitacions de què no es disposa d'un artista ni de material específic de qualitat, ni el temps que a un li agradaria poder dedicar, aquesta part s'ha encarat amb molta motivació i s'ha fet el millor possible tenint en compte que s'ha deixat la part artística més pel final per poder dedicar-se més a resoldre problemes de mecàniques i tecnologia.

Tots els gràfics són generats per l'autor mateix del videojoc, o recollits d'internet amb llicència gratuïta per a la seva utilització o extrets dels *assets* de llire utilització propis d'Unity.

5.2 GAMEPLAY

El *gameplay* és la font de l'entreteniment en tots els videojocs, és el més important i la primera cosa que hem de tenir en compte a l'hora de dissenyar un videojoc.

Les mecàniques són l'element que defineixen el *gameplay* del nostre videojoc. El *gameplay* consisteix en els reptes que el jugador ha d'encarar i les accions que el jugador pot realitzar per tal de superar-los.

5.2.1 DEFINICIÓ DE REPTES

El repte principal és una cosa tan general i simple com guanyar la partida, i d'aquest en surten diferents subconjunts de reptes que amb les accions del jugador es poden superar per assolir l'objectiu principal. Després ho veurem d'una forma més entenedora gràcies a la jerarquia de reptes, on veurem els diferents tipus de reptes des de el més simplificat de tots el qual en direm reptes de nivell atòmic (no divisibles) fins als de les capes superiors anomenats no-atòmics (formats per diferents reptes).

Els reptes del nostre videojoc són tots explícits, és a dir els jugadors el coneixen des de el principi, per exemple la condició de victòria, que és eliminar els personatges del jugador contrari, l'entretingut és que un mateix repte pot ser superat mitjançant diferents accions, que és la gràcia del joc, poder fer estratègies per arribar a la victòria de moltes maneres diferents.

A l'hora de fer els reptes, hem de posar-nos en la pell del jugador. Battleland és un joc que té temps per cada torn, i s'ha de saber combinar amb els reptes molt bé perquè el jugador no es frustri o s'estressi per culpa del temps. Com més curt és el temps, la situació es torna més estressant, és per això que Battleland atorga al jugador bastant temps per prendre decisions, ja que volem que sigui un joc d'estratègia en el qual el jugador tingui temps de planejar fins als següents moviments que farà i anticipar-se al rival.

En general, el tipus de reptes que apareixen en Battleland són del tipus de lògica i matemàtica, perquè hem de calcular danys, potenciacions, rangs d'atacs, paràboles, etc.

5.2.2 JERARQUIA DE REPTES

Com he dit anteriorment, el repte és molt clar, i la jerarquia que el descriu així ho demostra. La gràcia com s'ha dit anteriorment és que hi ha moltes maneres d'aconseguir superar els reptes atòmics. A el diagrama 1 podem veure el funcionament.

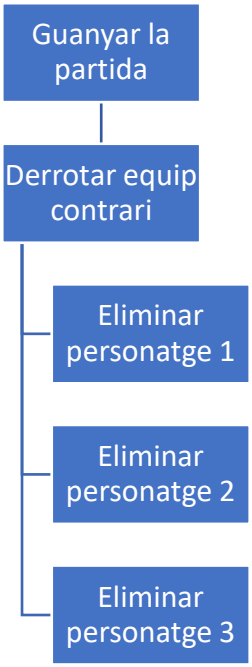


Diagrama 1 Jerarquia de reptes

Per exemple, per derrotar un personatge podem recórrer a diferents tipus d'estils de joc. En el diagrama 2 podem veure els diferent tipus d'accions que es poden dur a terme per a superar un repte.

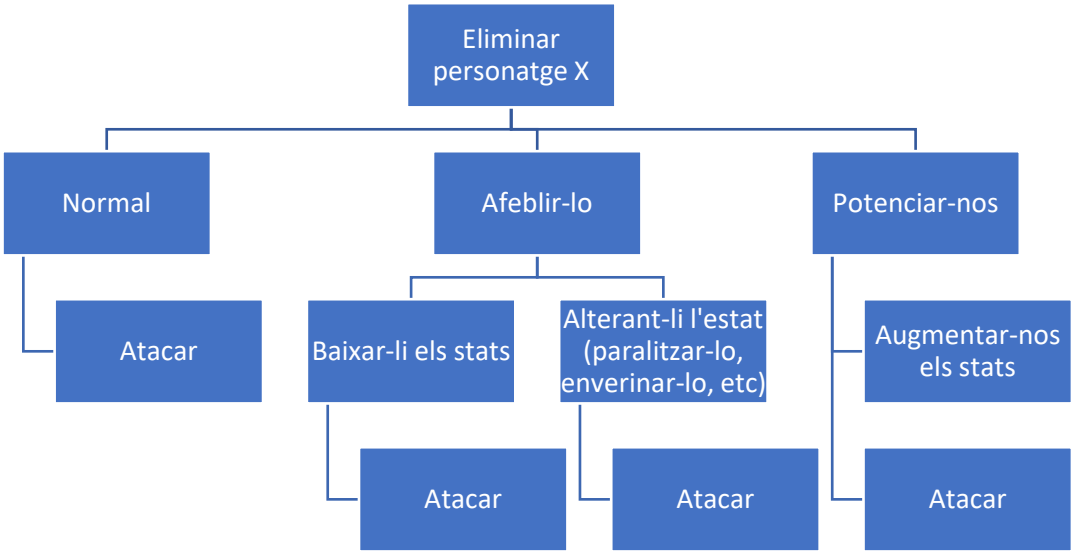


Diagrama 2 Jerarquia del repte 'Eliminar personatge X'

5.2.3 DEFINICIÓ D'ACCIONS

Són com bé diu el nom les accions que un jugador pot fer en el món del joc, com poden ser saltar, atacar, parlar, etc.

Per definir-les, ens hem de basar en els reptes que hi ha en el joc, principalment en els de nivell inferior o reptes atòmics, i hem de pensar en l'acció que permetrà superar cadascun d'ells. Hem de considerar que un repte pot ser superat per diferents accions.

També hi ha accions que normalment no van associades a cap repte, que poden ser accions per interactuar amb l'història del joc o el món del joc, accions per controlar l'estat del joc com per posar pausa, etc.

En Battleland hi ha les següents accions per superar els reptes:

- Moure's
- Atac bàsic
- Habilitats
    - Ofensives
    - Defensives
    - Ajudants
- Passives
    - Ofensives
    - Defensives
    - Ajudants

Per exemple, associant les accions amb cada repte, veiem que en el cas d'afeblir l'enemic, l'acció corresponent seria amb les habilitats ajudants, que dins d'aquestes hi hauria moltes possibilitats. Per exemple, un personatge pot paralitzar, un altre pot enverinar, un pot baixar-li l'atac, etc.

Amb les passives passa igual. Pot ser el cas d'un personatge que té com a passiva que quan mor explota. Llavors aquesta serviria per superar el repte d'eliminar l'enemic i seria una acció passiva-ofensiva.

5.3 SAVING MODES

Els jocs d'aquest tipus, que es basen en el multijugador Online, solen guardar tota la informació en servidors al núvol. L'aspecte negatiu d'aquests jocs és que només es pot jugar quan tens connexió.

Figura 26 Molts jocs requereixen Internet per jugar

En tenir el joc tota la informació important emmagatzemada en servidors al núvol, sumat al tipus de joc, provoca que la millor opció possible de guardar l'estat del joc sigui de forma **automàtica** al realitzar un canvi en les dades.

- Com són partides curtes i que no es poden aturar i reprendre en un altre moment (són partides en temps real contra altres jugadors en temps real), no té sentit un mode de pausa i permetre al jugador guardar en el moment que a ell li convingui.
- Tampoc se li pot permetre decidir si vol guardar o no, una vegada s'ha fet una compra o s'ha perdut una partida, encara que el jugador no estigui satisfet o content amb el resultat, no pot tornar a un estat anterior de joc perquè llavors no tindria sentit tenir rànquings i perfils de jugador.

5.4 MECÀNIQUES

No és molt diferent de les normes o instruccions que podem trobar per exemple en un joc de taula. El que cal saber sobre les mecàniques és que generen el *gameplay*:

* Defineixen els tipus de reptes que el videojoc pot oferir.

* Defineix les accions que el jugador pot fer per superar aquests reptes.

* Determina l'efecte de les accions del jugador envers el món del joc.

* Determina les condicions per aconseguir els objectius del joc i les conseqüències de tenir èxit o fallar en aconseguir-los.

Bàsicament, són les dades i algoritmes que en un joc de taula, defineixen les lleis del joc i les operacions internes. Però no hem d'oblidar que ara estem dissenyant, no ens hem de preocupar en com farem les coses, justament dictar-les.

Hi ha mecàniques que responen a reptes passius, per exemple moltes habilitats passives dels personatges son activades quan succeeixen certes condicions durant la partida. Per exemple, l'habilitat passiva del soldat és que quan aquest mata a un personatge enemic, el seu atac es veu augmentat per 2 unitats.

Hi ha moltes mecàniques que són de forma general en molts recursos del joc, són anomenades *reuse mechanics*. Per exemple, l'atac bàsic o el moviment. Tots els personatges sense excepció comparteixen la mecànica de moure's i d'atac bàsic, però tenen diferents paràmetres associats a ella. La reutilització de mecàniques és una eina bàsica per a poder generar nous reptes o noves accions i crear diferents variants.

5.4.1 RECURSOS

A continuació s'explicarà els recursos que s'han dissenyat pel videojoc. Els recursos vindrien a ser el que en programació entenem com a classes. Són la definició d'objectes del joc o materials que es poden crear, utilitzar, destruir, etc.

Taula 6 Recurs 'Personatge'

Personatge
Nom
Vida
Atac
Rang d'atac
Distancia recorrible
Habilitat
Passiva

Taula 7 Recurs 'Atac'

Atac
Dany
Distancia
Angle
Executor

Taula 8 Recurs 'Passiva'

Passiva
Nom
Efecte

Taula 9 Recurs 'Habilitat'

Habilitat
Nom
Torns necessaris
Efecte

Taula 10 Recurs 'Partida'

Partida
Escenari
Llista de *personatges*
Torn
Temps

Dels que s'ha detallat a continuació, veiem que la majoria són recursos tangibles perquè podem identificar-les en una partida i tenen propietats físiques en el món. Els recursos intangibles serien Partida, que representa les dades corresponents a l'estat del joc, i recursos que té el personatge com a atributs com són vida, distància recurrible, etc.

Hi ha recursos que són l'habilitat i la passiva, que són difícils de definir perquè poden variar molt entre uns i altres. Com s'ha explicat abans, poden haver habilitats ofensives, defensives o ajudants, o fins i tot habilitats compostes per diferents efectes, pel qual tindran paràmetres molt diferents entre si.

5.4.2 ENTITIES

Els *entities* són bàsicament la instància d'un recurs o l'estat d'algún element del món del joc. Per exemple, si tenim el recurs "personatge", podem tenir el *entity* "Soldat" tal que així.

Taula 11 Entitat 'Soldier'

Recurs : Personatge	Entitat
Nom	Soldier
Vida	10
Atac	*4*
Rang d'atac	20
Distancia recorrible	10
Habilitat	Granada
Passiva	Augment de dany al matar

Taula 12 Entitat 'Augment de dany al matar'

Recurs : Passiva	Entitat
Nom	Augment de dany al matar
Efecte	+2 d'atac
Disparador	Matar un personatge enemic

Taula 13 Entitat 'Granada'

Recurs: Habilitat	Entitat
Nom	Granada
Torns necessaris	1
Dany	7
Àrea	8

Com veiem, tenim entitats compostes, com són la passiva i habilitats que tenen diversos atributs. I després tenim una entitat per sobre que està composta d'altres entitats, com

és el personatge. I aquí es pot veure el que es comentava anteriorment de què és difícil definir els recursos habilitats o passives, perquè cada habilitat pot necessitar uns atributs diferents.

En un mateix equip, els personatges esdevenen una entitat única, perquè hi ha la restricció de què no es poden repetir personatges en un equip. El mateix succeeix amb l'entitat partida, que és únic per cada instància de batalla que es faci. En canvi, altres entitats com els atacs bàsics, etc., apareixen molts cops durant una partida.

5.4.3 DESCRIPCIÓ DE MECÀNIQUES

Fins ara s'ha descrit les diferents mecàniques que governen el joc de forma general, explicant els elements que trobarem en una partida i les seves característiques,

En aquest capítol es descriuran les diferents mecàniques del joc detalladament. Poden aparèixer mecàniques així com característiques que han sigut dissenyades pel producte final, però que en la versió de prototip no han sigut incorporades.

5.4.3.1 Objectius

Es tracta d'un joc simètric; tots els jugadors juguen amb les mateixes normes i intenten aconseguir la mateixa condició de victòria. Es tracta de què els jugadors tinguin les mateixes oportunitats per alçar-se amb la victòria i se sentin d'aquesta manera. L'objectiu del jugador en cada partida és el mateix, eliminar els personatges de l'equip contrari, i serà el propi jugador que hagi de pensar diferents combinacions i estratègies per aconseguir-ho.

5.4.3.2 Recompensa de final de partida

Al final de cada partida, depenent del resultat el jugador obtindrà diferent recompensa.

- Victòria: aconseguirà un nombre determinat de punts que se sumaran a l'indicador d'experiència del jugador. També aconseguirà un nombre determinat de monedes per a poder utilitzar després fora de batalla.
- Derrota: es restarà un nombre determinat de punts a l'indicador d'experiència del jugador. No aconseguirà monedes.

5.4.3.3 Condició de victòria

Un jugador guanya una partida quan aconsegueix eliminar tots els personatges de l'equip contrari o bé quan s'arriba al límit de temps té més personatges vius al camp de batalla que l'adversari.

També es considera com a victòria si el jugador contrincant es desconnecta de la partida en meitat de l'acció o si es rendeix.

5.4.3.4 Controls bàsics

El personatge es pot moure mitjançant els botons virtuals que apareixen a la pantalla en el costat esquerre inferior quan és el teu torn d'actuar, com es veu a la figura 27.

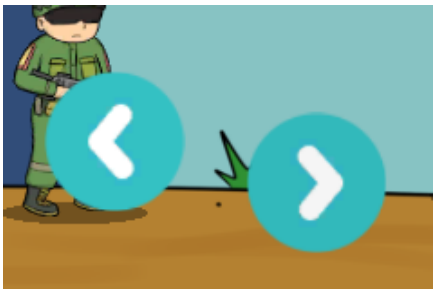


Figura 27 Controls de moviment

Com podem veure, només tenim disponibles el moviment en l'eix horitzontal, tot i que en un futur podria haver personatges que la seva habilitat fos volar per exemple i els permetés moure també en l'eix vertical. El moviment com està explicat té un límit de distància que es pot recórrer en un sol torn. Aquests botons poden ser presos clicant a sobre amb les mans si es tracta d'un dispositiu tàctil i clicant a sobre amb el ratolí si es tracta d'un dispositiu no tàctil.

D’altra banda, en el costat inferior dret apareixen els botons d’acció (Figura 28). El primer serveix per activar l’atac bàsic i el segon per activar la nostra habilitat especial. El botó d’habilitat especial no sempre està activat, ja que depenen de l’habilitat aquesta pot necessitar un nombre de torns per estar disponible.

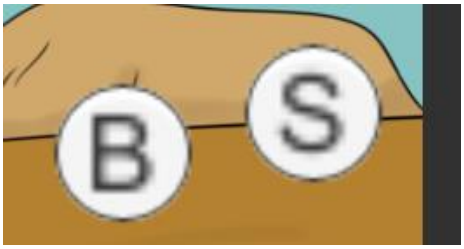


Figura 28 Controls d'acció, la B és per fet l’atac bàsic i la S per activar l’habilitat

Una vegada estem conformes amb la posició en la qual estem, podem clicar sobre el botó d’atac bàsic, quan estem apuntant amb l’atac bàsic, no podem moure el personatge.

Quan estem apuntant automàticament se’ns canvia els botons de moviment horitzontal i apareixen tres botons nous. Amb el primer botó podem tornar a l’estat anterior i tornar a moure el personatge. Amb els botons direccionals en l’eix vertical podem apuntar a quina posició volem disparar el projectil. També ens apareix una fletxa per indicar en tot moment quin angle estem agafant. En aquest estat també se’ns desactiva l’opció d’habilitat, i se’ns queda únicament el botó d’atac bàsic, que en tornar a ser premut, dispararà el projectil en el sentit de la fletxa. A la figura 29 podem veure un exemple del comentat en aquest paràgraf.

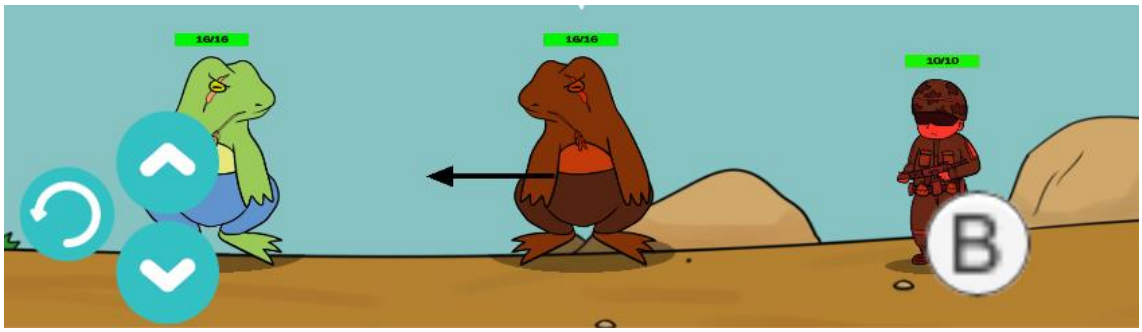


Figura 29 Captura de Battleland en el moment de fer un atac bàsic

I dels botons principals finalment tenim el d’habilitat especial. Aquest botó es diferent depenent del personatge, ja que com tenen diferents habilitats, la forma d’executar-les també varia.

Figura 30 Captura de Battleland: Soldat apuntant amb habilitat activada

En el cas de la figura superior (Figura 30), l'habilitat del soldat de guerra és llançar una granada, pel qual veiem que la mecànica de control és semblant a la de l'atac bàsic. També veiem que podem tornar enrere si no estem convençuts de voler utilitzar l'habilitat especial i que per utilitzar-la cal tornar a activar el mateix botó.

Com veiem a la figura 31, també tenim un petit botó sempre disponible el qual prement-lo, se'ns canvia de mode de càmera i podem moure la càmera lliurement, si tornem a premer-lo, torna al mode normal en el qual enfoca al personatge o objecte el qual està en acció.

Figura 31 Botó de canvi de mode de càmera

Tot el que és fora del camp de batalla, es basa en menús molt simples i sempre amb botons, pel qual el moviment entre pantalles sempre és molt intuïtiu i es basa en clics de ratolí o tocs a la pantalla si es tracta d'un dispositiu tàctil.

Figura 32 Captura de pantalla de Battleland: pantalla inicial

5.4.3.5 Mecàniques d'alteració d'estats

Hi ha moltes mecàniques que ocorren com a conseqüència d'una altra. Els personatges ,gràcies a les habilitats especials, habilitats passives i objectes poden donar la volta a la partida i canviar l'estil de joc dels personatges en temps de joc. Hi ha moltes funcions que permeten això:

- increment o decrement d'atac: un personatge que en principi té un atac baix i no es té en compte per l'enemic, pot tornar-se una basa ofensiva si es crea un entorn que afavoreixi el personatge i se li augmenta les estadístiques. A l'hora, un estil de joc d'afeblir el rival, pot provocar que el principal realitzador de dany de l'equip contrari quedi desfavorit per culpa d'habilitats i s'hagi de mantenir a cobert.
- curació de vida: tenir un personatge que cura pot provocar que puguis arriscar més del compte o apropar-te amb unitats inesperades.
- increment o decrement del rang d'atac: Si per exemple l'enemic té una unitat que proporciona molt dany i a sobre ataca a molta distància, pel qual pot cobrir-se bé les esquenes, és alterat per un decrement de rang, força al personatge a sortir de la seva guarida i quedar descobert. Passa similar amb el rang de recorregut.
- Paràlisis: Probabilitat de quedar-se el torn sense poder atacar. Jugar amb l'atzar.
- Verí, foc, etc.: Habilitats que prenen vida a poc a poc, o baixen les estadístiques.

5.4.3.6 Experiència i nivell

En una partida no hi ha nivell ni experiència, els personatges no evolucionen en el transcorregut del videojoc, a no ser que la passiva ho indiqui:
Per exemple, un personatge que comença sent molt fluix, però per cada torn que passa, se li suma un punt d'atac.
Tampoc es poden comprar millores en el joc perquè els personatges siguin millors que els altres.
Quan parlem d'experiència en Battleland ens referim a l'acumulació de coneixement que té un jugador. Els jugadors tindran un perfil i aquest contindrà una barra

d'experiència. Com s'ha dit anteriorment aquesta barra d'experiència s'omple a mesura que es juga i serveix per equilibrar el joc competitiu.

Aquesta barra també serveix per determinar el nivell de jugador. És un estimulant per als jugadors, per veure fins a quin nivell poder arribar i comparar-se amb amics o els millors jugadors del món. També s'ofereixen recompenses quan arribes a certa experiència i augmentes de nivell. És una motivació pel jugador fàcil d'implementar i que provoca que el jugador no vulgui quedar-se enrere envers els altres jugadors.

5.4.3.7 OBJECTES

En el joc, cada un nombre determinat de torns, apareix un objecte en el mapa de forma semi-aleatòria (s'estableix un mínim i màxim de torns entre els quals pot aparèixer l'objecte). Aquesta posició en el mapa sí que s'estableix totalment aleatòria. Hi ha diferents objectes i cadascun té una particularitat única, però mai tenen una finalitat ofensiva, sinó que són més bé potenciadors o ajudants.

Llista d'objectes

Objecte	Imatge	Descripció
Poma		Cura 5 punts de vida
Prismàtics		Augmenta el rang d'atac en 5 unitats durant 5 torns
Cinturó de karate		Incrementa l'atac en 3 unitats durant 3 torns

Idea futura: objectes equipables

En un futur i com a part d'una gran actualització, s'ha pensat en un sistema innovador que trencaria tot el *meta-game* [11] i canviaria moltes coses i donaria més hores d'entreteniment al videojoc.

Es tracta d'objectes equipables, els jugadors podrien intercanviar monedes de joc per objectes que es poden incorporar a cada personatge, i que afectarien als paràmetres únics de cada un.

Per exemple, es pot fer un objecte que sigui unes ulleres, i fan que augmenti el rang d'atac durant tota la partida al personatge que ho tingui equipat. Aquests objectes també podrien tenir efectes adversos així com les passives i fer que hi hagi molta varietat d'opcions.

Considero que és una gran idea per donar un gran cop de volta al joc, anunciant una nova temporada o macro-actualització del joc. D'aquesta forma, es podria fer que cada personatge pogués tenir molts més estils de joc i multiplicar exponencialment les estratègies. Per exemple, a un personatge ofensiu, se li podria equipar un objecte que li dóna més punts de vida i seria un personatge que aguanta molt i també fa mal, o se li podria incorporar un objecte que li augmenti encara més l'atac i fer-lo una amenaça permanent.

5.4.3.8 Funcionament d'una partida

El joc, sigui en local o online, té el mateix funcionament.

Es podria dir que es comença a jugar abans d'entrar a la partida, ja que l'organització de l'equip i els seus combatents es fa fora del camp de batalla, en una pantalla accessible des del menú on podrem veure els personatges i les seves característiques i llavors podrem crear formacions de batalla.

Una vegada els equips estan preparats i en el camp de batalla, aleatòriament el sistema elegeix l'equip que comença primer. L'objectiu és acabar amb els personatges de l'enemic o tenir més personatges vius que el contrincant una vegada termini el temps.

[11] *Meta-game* s'utilitza per indicar quines estrategies dominen el joc.

Hi ha un límit de temps total, si s'arriba a aquest temps i no hi ha jugador que compleixi les condicions de victòria, la partida es terminarà en forma d'empat. Tot i això, quan s'arriba a un temps determinat, la partida introdueix noves característiques perquè la partida sigui més emocionant, com per exemple que cauen meteorits.

5.5 FLOWCHART

El *flowchart* és un tipus de diagrama que representa un procés o un flux de treball. En el diagrama número 3 podem veure el flowchart del joc Battleland, ensenyant les diferents pantalles que hi ha hi com s'accedeixen a elles.

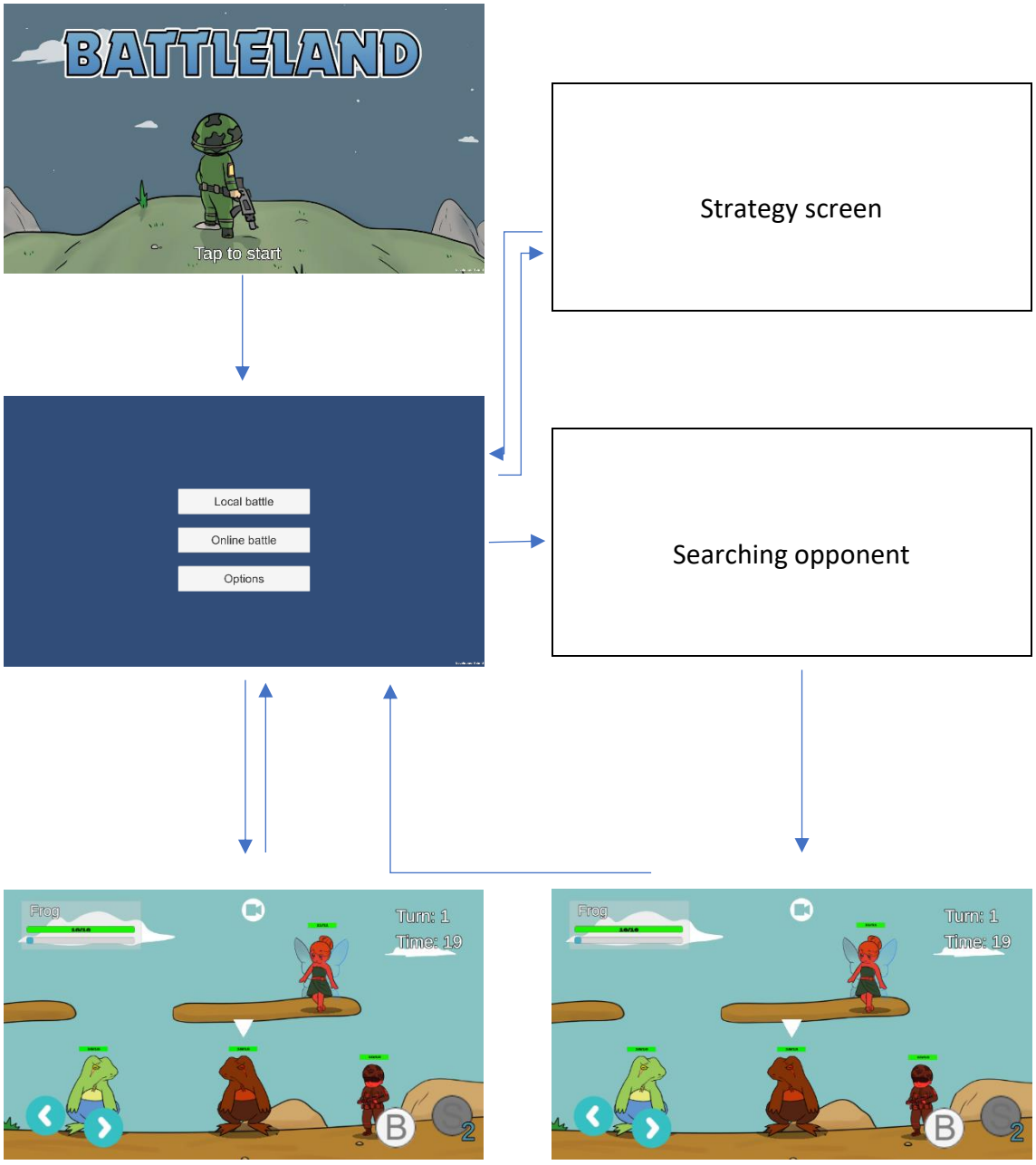


Diagrama 3 flowchart de Battleland

5.6 LEVEL DESIGN

Normalment, quan es parla de disseny de nivells la gent pensa en una successió de nivells en les quals s'ha dividit un joc o una història. En aquest cas, pel disseny de nivells bàsicament ens referim als diferents escenaris que trobarem dins el videojoc.

El disseny d'escenari és molt important, perquè pot dictaminar el tipus de reptes que podem trobar. Per posar-ho en context, podem trobar un mapa amb molta extensió, en la qual personatges amb molta mobilitat i molta distància d'apuntat tindrien avantatge, o podem trobar un mapa amb molts obstacles, pel qual certes habilitats com podria ser tirar una granada jugui en contra nostra i ens esclati a prop.

Per tot lo altre, els nivells tenen la mateixa configuració, però la idea és que cada vegada que s'introdueixi un personatge nou al joc, entri amb un nou escenari que mantingui l'ambientació del personatge, i per tant afavoreixi les seves habilitats.

5.5.1 ECONOMIA INTERNA

L'economia és el sistema pel qual recursos i entitats són produïts, consumits, i intercanviats en quantitats quantificables. És important en Battleland un equilibri que en videojocs s'anomena *static*, sent l'estat en què la quantitat de recursos produïts i consumits romanen sempre igual.

A continuació s'explica com els diferents elements del joc són controlats.

Fora de l'acció del joc

EXPERIÈNCIA

Cada perfil de jugador tindrà una barra d'experiència. Aquesta barra determina el nivell d'experiència del jugador, i és el criteri que s'utilitza per emparellar partides. La manera de guanyar experiència és guanyant partides i la manera de perdre-la és perdent-les. No té cap funció més, es podria dir que és el nivell del jugador.

MONEDES DE JOC

La manera de guanyar les monedes és guanyant partides, i en un futur es podria incorporar una altra característica d'obtenció de monedes per superar reptes o com recompenses per la fidelitat del jugador. Aquestes monedes no afecten el *gameplay*, ja que només es poden invertir fora del camp de batalla. Amb aquestes el jugador podrà comprar-se nous personatges, *skins*[12], o objectes pels personatges amb mecàniques *traders*.

També, en ser un videojoc que es pot descarregar gratuïtament, hi hauria una petita tenda on es podria comprar monedes de joc a canvi de diners reals per a poder complir els objectius empresarials i poder donar manteniment i nous continguts al videojoc.

Dins del joc

L'economia dins del camp de batalla recau sobre la quantitat de vida actual, l'atac, la distancia que pot recórrer per torn, i el rang de l'atac bàsic de cada personatge.

VIDA

Abans de començar a explicar la economia de la vida, cal remarcar que hi ha dos variables, la vida màxima del personatge, que és la vida amb la qual comença i la qual estableix el límit de vida que pot tenir el personatge, i la vida actual, que és la vida que té el personatge en el moment de l'acció. Quan la vida actual d'un personatge arriba a 0, a no ser que hagi alguna habilitat que indiqui el contrari, el personatge en qüestió morirà. La manera de perdre vida és lògica, si t'impacta un atac enemic, et restarà a la teva vida actual el dany de l'atac. També podria haver un personatge que la seva habilitat o la passiva impliquessin prendre vida al mateix personatge a canvi d'altres beneficis. De igual manera, la vida actual es pot guanyar gràcies a habilitats o passives de personatges. També es pot guanyar vida agafant algun objecte del mapa que tingui aquesta funció.

DANY

De la mateixa manera, cada personatge comença la partida amb una quantitat de dany establert per al seu atac bàsic. Aquest atac pot ser incrementat o disminuït depenent d'habilitats del mateix personatge o altres personatges, així com passives. Així com la

[12] Aspectes diferents pels personatges

vida actual té un límit que estableix la variable vida màxima, amb l'atac no passa el mateix, i si les circumstàncies de la partida ho permeten, es podria incrementar indefinidament l'atac, per contra, no es pot restar atac fins que tingui un valor negatiu. També es pot incrementar l'atac recollint objectes del mapa que tinguin aquesta funció.

RANG D'ATAC I DISTÀNCIA

Igual que passa amb l'atac, tots els personatges tenen definits més paràmetres únics, com el rang de l'atac bàsic i la distància que pot recórrer el projectil. Aquests poden ser modificats el seu valor gràcies a habilitats de personatges aliats o enemics o a conseqüència d'alguna passiva. En aquest cas, és necessari establir una limitació, ja que si no un personatge que li hagin deixat a 0 el rang d'atac es podria quedar diversos torns sense poder fer res. També es pot alterar aquests paràmetres recollint objectes del mapa que tinguin la determinada funcionalitat.

OBJECTES

Els objectes són generats a partir d'un determinat torn i, a partir d'aquest el rati de producció és variable. La font d'aquest recurs és el controlador de joc perquè és qui controla en tot moment els torns, etc. No hi ha un *spawn point*[13] concret, la idea és que per a què sigui el màxim de just possible, els objectes són generats de manera semi-aleatòria al voltant de tot el mapa. La idea es que no apareguin més de 3 objectes per partida. I els objectes també s'escullen de forma aleatòria, assignant una llista de possibles i escollint un d'ells a l'atzar.

Font com a mecànica

En un futur podria haver-hi personatges que generessin *entities*. Per exemple, un invocador que amb la seva habilitat crees monstres que l'ajudessin a lluitar. O un personatge que crees zones on els personatges que siguin a dins sofreixin algun estat alterat. També es podria fer una mecànica de tipus *converter*, per exemple una habilitat que canvies un percentatge de vida del personatge a canvi d'un augment de dany. Hi ha moltíssimes possibilitats.

[13] Posició des de la qual s'instancien objectes

5.6.2 BALANCEIG DEL JOC

En un joc d'aquestes característiques és molt important un correcte balanceig, perquè el més important en un joc d'estratègia és que el que determini el guanyador sigui la mateixa habilitat del jugador. És molt difícil equilibrar-lo perquè cada vegada que apareix un personatge, no només has mirar-lo a ell, ja que pot ser que amb la seva introducció al joc altres personatges siguin *buffats*[14] o *debuffats* indirectament. Per exemple, un personatge que incrementa el seu dany quan un personatge aliat mata a algun enemic, seria una bona combinació amb el soldat, que té una passiva que fa el mateix però per si mateix. Amb només una mort d'un personatge, dos en sortirien beneficiats. També hem de tenir en compte els diferents escenaris, el disseny de nivell és molt important per a no afavorir massa a unes unitats que a unes altres.

En aquest tipus de jocs, solen haver-hi moltes actualitzacions a l'any per fet petits canvis en el balanceig. És la forma de què unitats no es quedin desfasades i evitar el que es diu una estratègia dominant. El que es vol amb això és que el multijugador competitiu no sigui un avorriment perquè hi ha una formació predominant. El que s'ha comentat abans sobre que un personatge nou pot indirectament ajudar un altre personatge, també són coses a tenir en compte per balancejar el meta joc.

Per a què Battleland sigui el més just possible, s'intenta evitar el màxim possible l'atzar, quedant-se reduït a indicar l'equip que li toca jugar primer, i a algunes habilitats.

Encara que hi hagi habilitats que tinguin un component d'atzar, l'habilitat es balanceja igualment, per exemple, hi ha una habilitat del personatge "Frog" que a primera vista és molt bona, perquè a tots els personatges que toca els hi aplica possibilitat de paràlisis durant els dos següents torns. Per a què no sigui un abús, se li introdueix aquesta possibilitat de quedar-se o no paralitzat perquè sinó podria desnivellar massa el joc. També s'equilibra sent una habilitat que no provoca dany i que es necessiten dos torns per activar-la. És un dels molts exemples que poden haver-hi, així també es permet al jugador decidir si vol o no arriscar, i això forma part de l'estratègia de batalla i dóna molta emoció.

La gràcia de la varietat de personatges és que poden succeir molt tipus de situacions, per exemple, hi ha personatges que poden tenir un inici de partida molt potent i altes al

[14] *Buff* és el terme utilitzat quan un personatge rep una millora.

revés, pel qual sempre hi ha oportunitats de guanyar la partida. És molt difícil trobar una formació que sigui completament *counter* [15]de la teva, però és cosa dels jugadors que han d'analitzar que es el que es juga més i com contrarestar-lo. Hi ha unitats que són moltes fortes contra altres individualment, però per això els equips són de 3, per a poder fer combinacions que enforteixen les unitats. Per tant ens trobem que individualment hi ha mecàniques de RPS (rock-paper-scissors), que s'anul·len en el moment de crear els equips.

És important balancejar els personatges i el joc per a què les partides no siguin massa curtes, llargues o, fins i tot infinites. Per exemple, si l'habilitat de curar cura més del dany que et fan, esdevé una estratègia dominant i pot fer que una partida no acabi mai si els dos equips ho tenen.

La progressió en la dificultat està ben construïda perquè com els jugadors juguen segons el seu nivell d'experiència, sempre es trobaran amb gent d'experiència similar, per tant la dificultat augmenta a mesura que augmenta la comprensió i habilitat del jugador. D'aquesta manera també maneguem la dificultat per a mantenir al jugador sempre entretingut el que es diu en *Flow State*; sense arribar a la frustració per jugar amb gent massa bona pel teu nivell o massa dolenta que et provoqui l'avorriment.

[15] Quan un personatge o estratègia domina envers una altra, es diu que es *counter*

5.7 HISTÒRIA

El videojoc no compta amb història. Només és una gran superfície on s'ajunten diferents personatges que busquen guanyar la batalla.

5.7.1 MÓN DEL JOC

L'univers en el què es basa Battleland, és un univers imaginari on hi ha tota classe d'éssers vivent en ell. L'estil *cartoon* així ho permet, i d'aquesta manera podem trobar des d'un humà com és el soldat, com una fada o fins i tot una granota que porta pantalons. Passa el mateix pels escenaris, podem trobar una ciutat, com una platja, com l'interior d'un volcà o lluitar en la mateixa lluna.

Els personatges tenen representació 2D, però amb els escenaris es busca crear sense sortir del 2D amb sensació de profunditat. L'espai dels escenaris es busca que sigui bastant petits, perquè volem que les partides siguin curtes, i que el jugador no es passi temps només caminant, sinó que des de el primer torn comenci l'acció. L'espai és molt important perquè una unitat de distància pot ser un factor decisiu per decidir una partida.

No hi ha un temps en l'univers, en ser un món fictici, no ocorri ni en el passat ni en el futur. No és una cosa important ja que el joc no consta d'una narrativa com a tal.

En conclusió, veiem que el nostre joc no és necessari aprofundir massa en la dimensió física ni temporal, perquè no forma part del *gameplay*, sinó que és un component purament cosmètic.

5.8 INTERFÍCIE GRÀFICA

En aquest capítol s'explica el conjunt d'escenes que té el videojoc i també es detalla els elements més importants que veiem en el camp de joc.

Primer, començarem amb les escenes explicant-les una mica i tot seguit veurem els elements de la interfície gràfica.

Pantalla inicial

Figura 33 Pantalla inicial

Aquesta és la primera cosa que veiem quan entrem al joc (després del logotip d'Unity, que en la versió gratuïta és obligatori que aparegui).

És simplement una pantalla per a ensenyar el títol del joc, i per a què el jugador no entri directament en un menú. (Figura 33)

En clicar s'espera un petit temps i llavors dóna accés a la pantalla de menú principal.

Menú principal

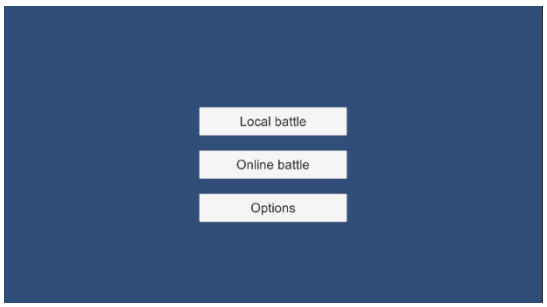


Figura 34 Menú principal

En el menú principal, el jugador pot seleccionar a quin mode de joc vol jugar, o accedir a la pantalla de formació d'equips o a les opcions.

Consta de tants botons com escenes a les quals es pot accedir, s'ha volgut buscar la senzillesa.

Deixarem pel final les pantalles de batalla perquè s'aprofitarà per comentar els elements de la interfície.

Pantalla estratègia

En la pantalla estratègia és on el jugador pot veure els seus personatges tant desbloquejats com bloquejats, i pot accedir a la informació de cada personatge. Quan cliques sobre un personatge, en lloc de canviar d'escena, s'obre un petit pop-up on apareix *l'sprite* [16] del personatge juntament amb les seves dades bàsiques i una explicació de les seves habilitats especials i passiva.

En aquesta pantalla podem escollir els personatges que formaran part de l'equip. Es poden crear diferents equips, i amb monedes de joc es poden comprar encara més espais.

L'equip que es deixa en última instància és amb el que competirem en la següent partida.

[16] Els *sprites* són mapes de bits que representen gràfics del joc.

Opcions

En el menú opcions, hi haurà diferents configuracions que podem modificar al nostre gust. D'aquesta manera podem personalitzar el videojoc de la manera que ens vagi millor. Hi haurà opcions bàsiques, modificar el nivell de volum, l'idioma del joc, i diferents aspectes de configuració.

Camp de batalla

El camp de batalla ja l'hem pogut presenciar en diferents seccions, però ara ens dedicarem a comentar els diferents elements de la interfície.

Figura 35 Camp de Batalla

Quan entrem a una partida trobarem sempre una situació similar a la que es veu en la imatge superior (Figura 35). S'intenta no sobrecarregar massa la pantalla, i posar només els elements més essencials per a poder entendre l'estat del joc. A continuació, i començant per la cantonada superior esquerra, comentarem un a un els elements de la interfície.

Figura 36 Informació de personatge

En aquest petit requadre podem veure la informació més bàsica del personatge del torn actual. En aquest podem veure; el nom del personatge, la barra de vida, i la barra de desplaçament del personatge. (Figura 36)

Figura 37 Informació de partida

A la cantonada superior dreta podem veure informació general de la partida, concretament el torn en el què estem i el temps restant que queda per terminar el torn actual. (Figura 37)

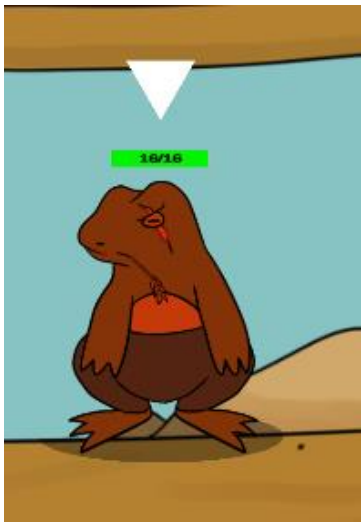


Figura 38 Personatge actual

A sobre de cada personatge també veiem la barra de vida.

Com es pot veure a la imatge superior, quan és el torn del personatge que correspongui, una fletxa apareix sobre la barra de vida per indicar clarament a qui li toca actuar. (Figura 38)

Figura 39 Botó especial

Al botó d'habilitat especial, quan aquest està desactivat apareix un petit indicador dels torns restants necessaris per a què l'habilitat estigui disponible per ser executada.

També hi ha altres elements que apareixen depenent de les circumstàncies, com per exemple quan estem apuntant que apareix una fletxa per ajudar-nos a saber la direcció en la qual sortirà disparat el projectil.

No hi ha opcions de pausa dins del joc per diversos motius, primer perquè són partides curtes i tenir un botó de pausa podria provocar frenar la dinàmica de joc, i també i més important perquè és un joc basat purament en el multijugador, i aquests jocs no solen tenir pauses perquè trenquen l'experiència de joc.

5.9 ART DEL JOC

Aquí apareixeran la majoria de materials utilitzats per a realitzar el videojoc. A continuació i per seccions per classificar-ho millor es mostren aquests materials.

5.9.1 ESCENARIS

A les figures 40 i 41 podem veure dos exemples d'escenaris utilitzats durant el desenvolupament del prototip.

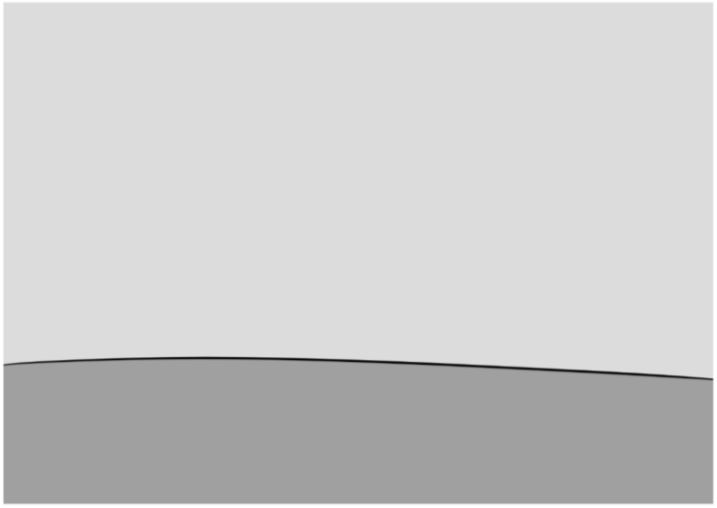


Figura 40 Escenari bàsic de proves

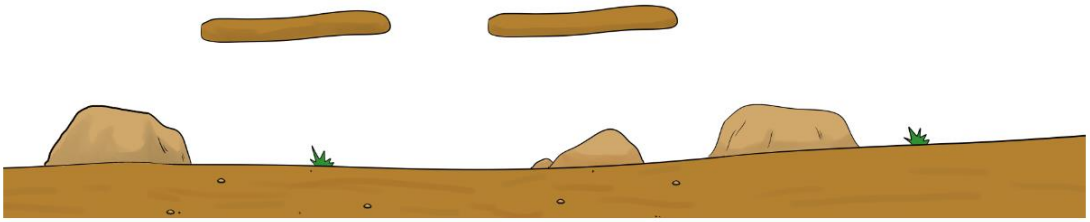


Figura 41 Escenari de proves una mica més treballat

5.9.2 PERSONATGES

A les figures 42,43 i 44 podem veure diferent art desenvolupat dels personatges.

Figura 42 Art del Soldier

Figura 43 Art de Fairy

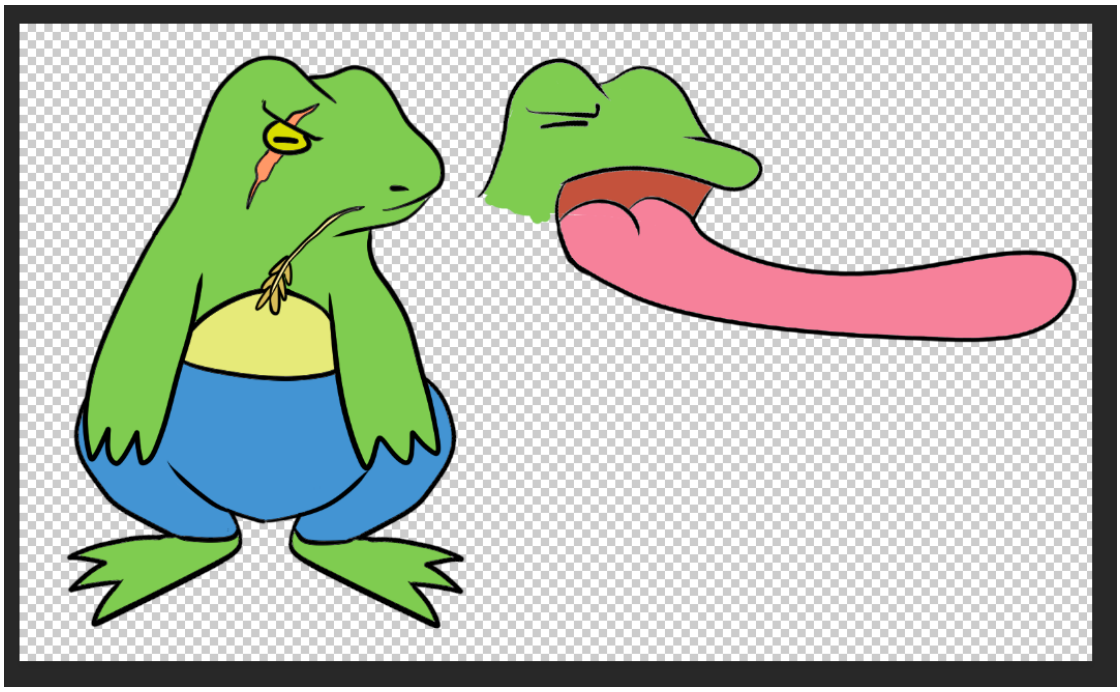


Figura 44 Art de Frog

5.9.3 EFECTES VISUALS

En aquest capítol es recopilen alguns efectes visuals lligats a diferents personatges, com poden ser a conseqüència de la utilització d'alguna habilitat o l'efecte d'alguna passiva. En la figura 45 podem veure un cas d'exemple.

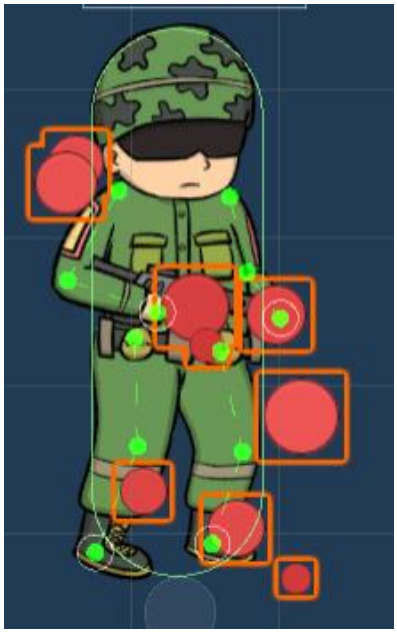


Figura 45 Partícules generades al ser disparada la passiva del Soldier

Per ensenyar que un personatge ha rebut una millora o també per demostrar el contrari, s'utilitzen partícules. En el cas que sigui una potenciació, les partícules pujaran des de baix cap avall. En el cas que sigui un deteriorament, aquestes aniran de dalt cap a baix. És una manera clara i entenedora que ja s'ha utilitzat en altres videojocs i que dóna a entendre la situació fàcilment. El color o forma d'aquestes també ajuda a entendre el seu efecte. Per exemple el verd s'associa a la cura, mentre el vermell al dany.

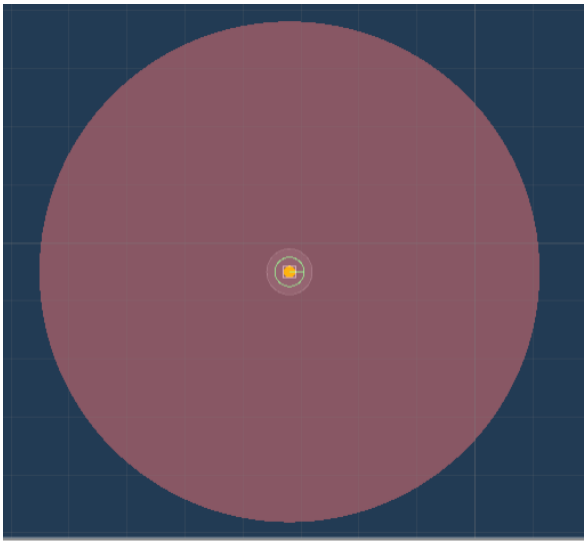


Figura 46 Granada (Habilitat Soldier)

Per veure el rang d'un atac, també es pot utilitzar combinació d'animació i partícules per crear un bon efecte visual. En el cas de la granada (Figura 46), el cercle rodó al voltant ens ensenya si estem dins del rang de perill de la granada o no.

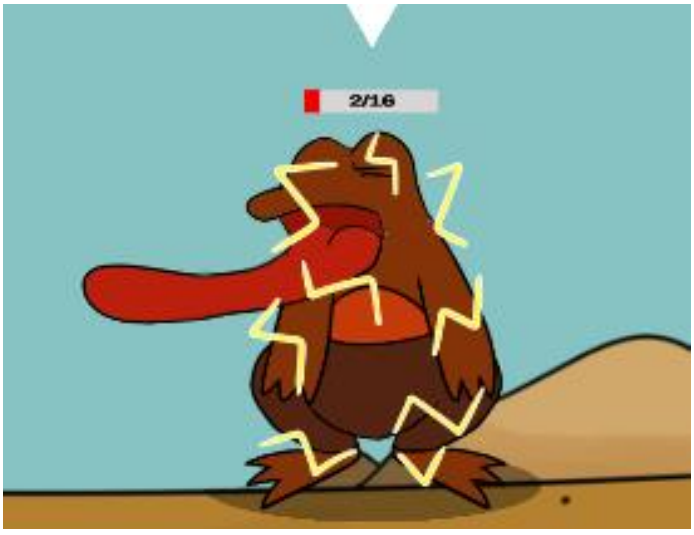


Figura 47 Quan quedes paralitzat apareixen uns raigs al voltant

Quan tenim un efecte advers, com pot ser paràlisis o enverinament, també apareixerà algun efecte visual per a situar-nos. En el cas que veiem en la imatge anterior (Figura 47), el personatge està paralitzat i apareixen uns raigs al voltant del cos. Cada efecte advers seria representat de manera diferent.

Figura 48 Fairy té a Frog seleccionat

Quan se selecciona un personatge, aquest canviarà de color per donar-ho a entendre. A la imatge anterior (figura 48) en tenim un exemple de quan la fada selecciona una unitat per curar-la amb la seva habilitat. S'ha utilitzat el color groc per representar-ho, ja que el vermell s'utilitza com a model alternatiu per diferenciar el mateix personatge en equips diferents.

5.9.4 EFECTES DE SO

El videojoc, com a mínim el prototip, no consta d'efectes sonors, ja que s'ha prioritzat altra feina abans. En tot cas, en no tenir les eines ni personal que tingui la capacitat per generar-lo, es disposaria a utilitzar sons i música de llicència oberta.

5.9.5 ANIMACIONS

En un principi, es va pensar pels casos següents en les següents animacions:

1. *Idle*: Quan el personatge està en repòs.
2. *Moving*: Quan el personatge s'està moguen (Figura 49)
3. *Aiming*: Quan el personatge està apuntant.
4. *Hability*: Quan el personatge utilitza una habilitat.
5. *Passive*: Quan és disparada una passiva.
6. *Injuried*: Quan perds vida.

Amb el primer personatge es va arribar a la segona animació, i es va comprendre que es necessitava molt temps per aconseguir uns resultats decents. És per això que es va decidir deixar les animacions pel final si hi havia temps.

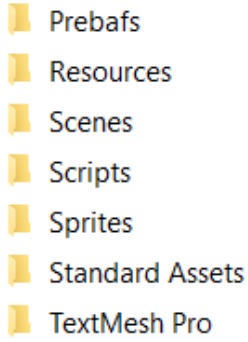
Figura 49 Animació del Soldier

6. IMPLEMENTACIÓ I PROVES

En aquest apartat es comentarà tot el que s'ha pogut generar com a demostració jugable del projecte. És un prototip per veure reflectida la idea en general, pel qual hi ha moltes característiques que no s'han arribat a desenvolupar i el que es veu no tindria res a veure amb la versió final. El que s'ha volgut plasmar sobretot és el *gameplay* d'una partida corrent amb uns quants personatges diferents per veure de què va el tema de les formacions. En aquest prototip també es pot veure una idea de com són les interfícies i el moviment entre menús. En aquest apartat no s'ensenya codi, aquest anirà adjunt en l'annex, és un apartat per entendre el disseny del *gameplay*.

A continuació es comenta diferents aspectes de la implementació i de com està estructurat el joc. Com es diu en un principi, el videojoc està format amb Unity i per tant l'estructura i el tipus de programació té un determinat estil propi d'Unity.

Per començar, explicarem l'estructura de carpetes del projecte.



S'ha dividit el tipus d'objecte tractat en diferents carpetes per a poder organitzar millor tots els *assets* i poder accedir més fàcilment a ells. S'ha creat carpetes per a les coses més essencials, com els *scripts* o els *sprites*, per exemple. Dintre d'aquestes també hi ha subdirectoris per a tenir-ho tot molt ben organitzat. Per exemple, dins de *sprites* tenim diferents carpetes per si es tracta d'un personatge o un altre, o són *sprites* d'objectes, etc.

A mesura que el projecte anés creixent potser es canviaria certes coses de l'estructuració per encara ordenar-ho millor, per exemple fent una carpeta només per les animacions, però al nivell de complexitat que s'ha arribat es considera suficient l'estructura aquí mostrada.

6.1 ENTORN

L'escena principal on passa l'acció és el camp de batalla. A continuació explicarem com està formada aquesta pantalla i quins elements la componen. A la figura 50 veiem la jerarquia principal d'objectes de l'escena i seguidament es comenten un a un.

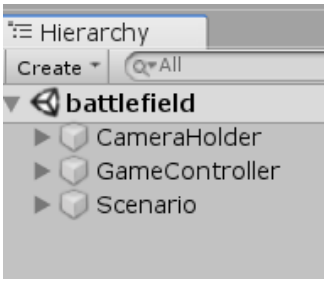


Figura 50 Jerarquia d'objectes de l'escena Battlefield

L'escena està formada per tres elements principals, un que seria el *CameraHolder,* el qual controla tot el que te a veure amb la càmera i els elements de la interfície, així com els controls virtuals. Després tenim el *GameController* que conté tota la informació dels equips i controla l'estat del joc. Finalment hi ha *Scenario* que és on està la informació del nivell.

Com veiem és un escenari senzill amb una gran plataforma de terra a sota de la pantalla i dues plataformes volants al centre. En el dibuix, s'ha fet un escenari bastant desèrtic en el que hi ha algunes roques i males herbes. (Figura 51)

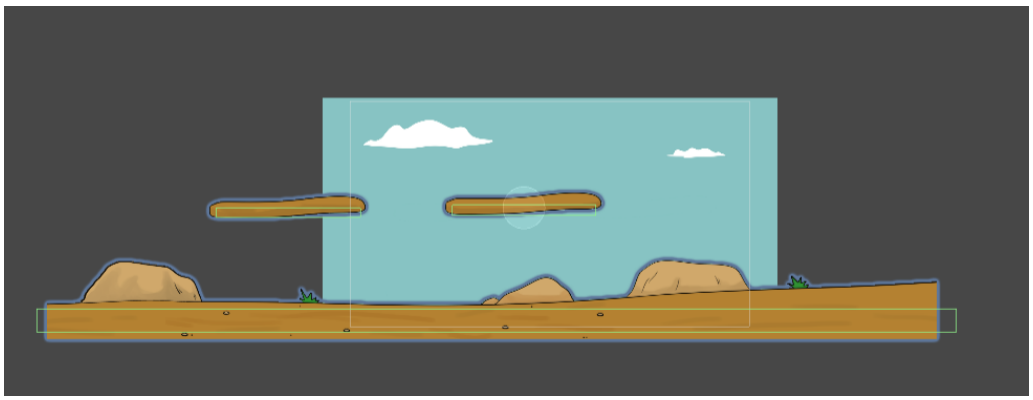


Figura 51 Escenari de proves

El gameobject que compon l'escenari, anomenat "Scenario", està format per un objecte "DieZone" el qual conté un *collider* pel qual si un personatge cau de l'escenari, aquest morirà en entrar en aquesta àrea. Seguidament tenim "floor", "platform 1 i 2", els quals són els objectes que contenen els *colliders* pels quals s'aplica la física i per tant el personatge podrà caminar sobre d'ells. Finalment hi ha "simple", que conté la imatge de la terra. La imatge que conté el cel no està en aquest grup, quan arribi el seu moment s'explicarà el perquè. (Figura 52)

Figura 52 Scenario

Després tenim el *gameobject* "GameController", que conté l'script que controla l'estat de la partida en tot moment. (Figura 53)

Figura 53 GameController

Dins d'aquest tenim els dos equips amb els dos equips, els quals a dins tenen els objectes dels personatges, que s'explicaran després. Finalment hi ha alguns elements que podrien estar en un altre lloc o podrien instanciar-se en temps de joc, però que com està programat de forma que el "GameController" els controla, m'ha semblat lògic que estiguessin junts.

Als equips, trobem els personatges que el componen. Aquests personatges estan formats de la següent forma (Figura 54)

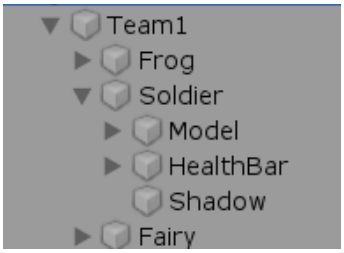


Figura 54 Team i Personatge

Tots els personatges estan fets de la mateixa manera, en aquest cas hem obert el *GameObject* corresponent al personatge "Soldier" per poder observar com està format. Primer, a l'arrel, està les dades importants del personatge com els *scripts*, els *colliders* i si és el cas pot haver-hi altres coses com el component de les partícules o el que sigui.
Després té 3 fills, un anomenat Model, que és on està l'art del personatge amb el seu esquelet i conseqüentment l'animador.
Després hi ha la "HealthBar", que és la barra de vida que tenen sobre el cap durant tota la partida.
I finalment, provisionalment es va generar una ombra creant un *sprite* semi-transparent com a fill de l'objecte, amb un resultat bastant convincent i ben simple.

Finalment hi ha el component que engloba la càmera i també el *canvas*[17], que conté les interfícies gràfiques. En el *canvas* podem veure tots els elements de la interfície, tant els botons, com els diferents camps de text, etc. L'objecte, anomenat CameraHolder està estructurar de la següent manera (Figura 55):

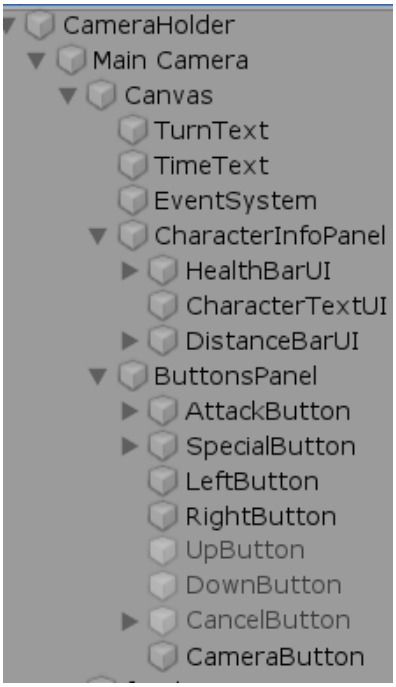


Figura 55 CameraHolder

[17] El canvas és el llenç on es dibuixen els elements de la interfície

6.2 IMPLEMENTACIÓ CÀMERA

La càmera de Battleland és una càmera "**intel·ligent".** Aquesta càmera sempre segueix l'acció del joc, sigui un personatge o un atac. Normalment el controlador de joc o l'objecte que sigui pot parlar-li a la càmera i dir-li "apunta'm a mi", però la càmera és capaç de buscar el personatge que ha de seguir, si porta gaire estona parada.

També s'ha fet que la càmera segueixi fins a uns certs límits del mapa, per el qual si el personatge s'apropa massa al límit esquerre aquesta s'atura.

Té un mode de càmera lliure, que en ser activat permet al jugador desplaçar-la al seu gust.

També s'ha cercat a la Internet i s'ha obtingut alguns scripts que fan que es pugui fer zoom amb controls tàctils, i un altre que fa que la càmera tremoli. Aquesta acció per exemple és cridada quan esclata una granada, creant un efecte immersiu.

Dels paràmetres de la càmera només és important que s'ha de marcar la opció *Ortographic* en l'atribut de projecció, i que el zoom es controla mitjançant l'atribut *size*. (Figura 56)

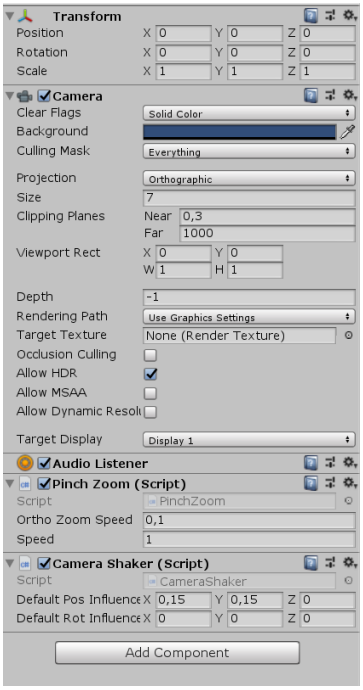


Figura 56 Càmera

Del *canvas* també és necessari indicar que es vol escalar els elements en referència a la mida de la pantalla, per així evitar que els botons es facin més grans o petits en canviar de resolució.

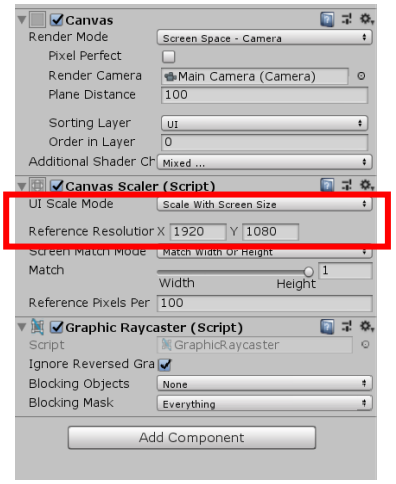


Figura 57 Canvas

6.3 IMPLEMENTACIÓ BOTONS

El controlador dels botons té un atribut que controla el personatge al qual li correspon moure, amb el que pot saber en tot moment el seu estat. Per això, és capaç de desactivar o activar els botons que faci falta depenent de si el personatge està apuntant, està en repòs, etc.

6.4 IMPLEMENTACIÓ PERSONATGES

Tots els personatges segueixen el mateix patró. Estan formats pel mateix tipus de *GameObjects* i els seus components també són iguals. A continuació s'explica amb més detall. (Figura 58)

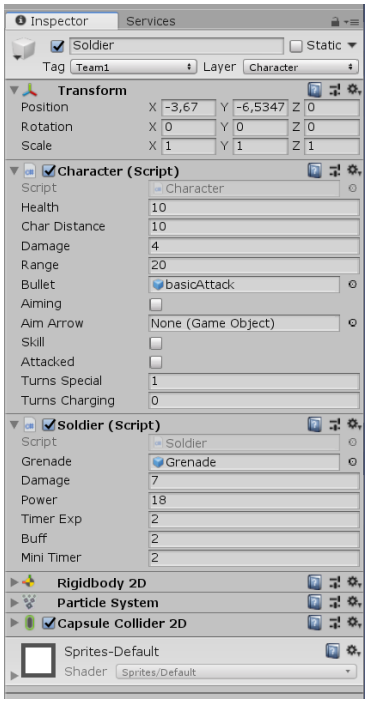


Figura 58 Personatge

Normalment un personatge està format per un *transform* (obligatori en cada *GameObject*), un *RigidBody* i un *Collider*. Després tots els personatges tenen un component "character", que és on estan els paràmetres bàsics del personatge com són la vida, el dany, els torns per tenir l'atac especial, etc. Finalment, cada personatge té un script únic que és el que conté les habilitats especials i passiva.

Està fet d'una manera que sembla que el personatge i les habilitats estiguin separats, però és pel fet que amb la forma de treballar d'Unity amb els components he preferit fer-ho així per a tenir el codi més organitzat. Encara que no està implementat com a tal, podem imaginar "Character" com una superclasse de la qual hereten els personatges específics com "Soldier", "Fairy" i "Frog".

El *script* "Character" conté la informació del personatge, i també té implementada les principals accions que poden executar tots els personatges. Aquestes són desplaçar-se pel mapa, fer l'atac bàsic, i també avisen per a què l'interfície canviï i s'actualitzi amb la nova informació de l'estat del personatge. Des de l'script també s'actualitzen els paràmetres que l'animador utilitza com a referència per canviar d'animació.

6.4.1 SOLDIER

El *GameObject* "Soldier" conté un script "Character" com s'ha explicat i després té un *script* anomenat "Soldier", que és on es controla les habilitats i les passives del personatge. Pel "Solider", que es vol que sigui un personatge equilibrat s'ha considerat aquests valors.

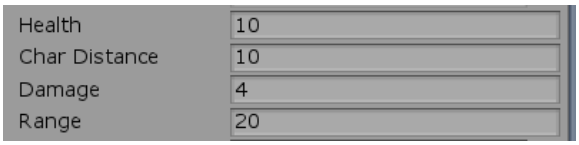


Figura 59 stats Soldier

Quan es clica al botó d'habilitat especial, *l'script* s'encarrega de posar el personatge a apuntar i de què quan es torna a clicar s'instanciï la granada en la posició i direcció que apunta el personatge.

Després té un comptador de les morts del personatge, amb el qual cada vegada que el "Soldier" mata a algun enemic, l'incrementa la força en dues unitats. A la figura 60 podem veure com la interfície s'encarrega de mostrar-li al jugador aquest efecte, apareixent un indicador de la força augmentada al costat de la barra de vida i instanciant unes partícules.

Figura 60 Passiva Soldier

6.4.2 FAIRY

La fada s'ha pensat com un personatge que juga a darrere dels personatges ofensius i per això té menys valor en punts com el dany. Com es pot veure a continuació, té més vida que el soldat, però en tot el restant és pitjor. (Figura 61)

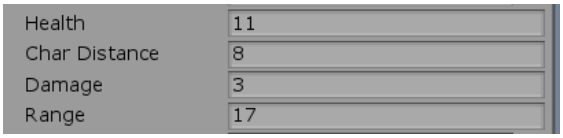


Figura 61 stats Fairy

Però, té unes habilitats moltsį interessants per a ajudar als restants membres de l'equip. La fada té un script anomenat "Fairy" que controla l'habilitat de curar i l'habilitat passiva auto-curativa. Quan s'activa permet a la fada seleccionar un personatge aliat i curar-lo. Per a això, es recull la posició del ratolí, i s'utilitza un *raycast* 2D, que surt des del centre de la càmera al punt i així podem veure quin *GameObject* travessa. A continuació es mira si és un personatge del mateix equip i si és el cas, se li canvia el color perquè es vegi que està seleccionat i llavors el cura 3 punts. Quan fa això també genera unes partícules en la unitat seleccionada.

La passiva fa que sempre en acabar el torn, faci el que faci, es curi a ella mateixa un punt.

Figura 62 Passiva Fairy

6.4.3 FROG

El “Frog” és un personatge que té la funció de ser el tanc i la primera línia en el camp de batalla, és per això que té molta més vida que els altres, però els seus altres estats són més baixos. (Figura 63)

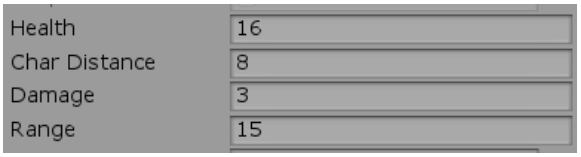


Figura 63 stats Frog

Tot i això és un personatge que pot ser la amenaça més gran, perquè la seva habilitat de paralitzar pot decidir una partida.

Quan s’activa l’habilitat, estira la seva llengua i tots els personatges enemics que siguin impactats per ella rebran l’estat “frogged”, el qual farà que tinguis un 50% de possibilitats de quedar-te paralitzat en atacar i a sobre et redueix a la meitat el rang d’atac i la distància que pots recórrer. Com es una habilitat molt potent, és l’única dels personatges del prototip que necessita dos torns per carregar-se.

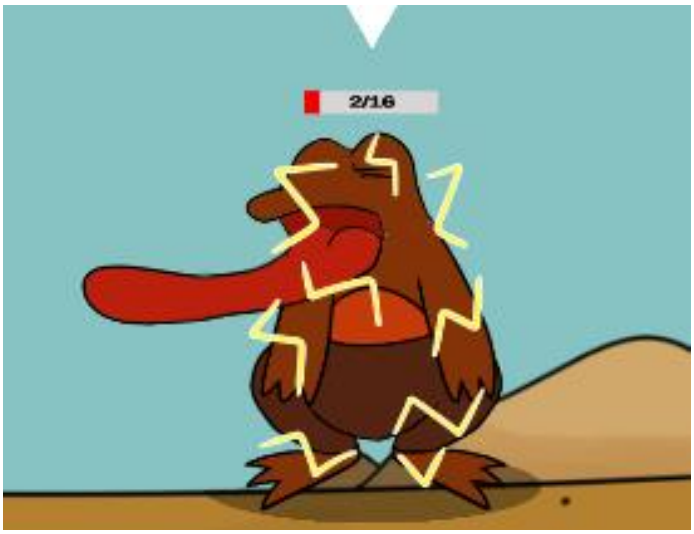


Figura 64 Estat "frogged"

6.5 ALTRES

6.5.1 HEALTHBAR

La barra de vida de cada personatge és un *GameObject* molt senzill que està lligat al personatge en forma de fill, pel qual sempre que es mogui el personatge es mourà amb ell.

Consulta la vida del personatge, i quan canvia el valor d'aquest, salta un disparador i es mira el percentatge de vida que li queda al personatge. Depenent de la vida, la barra canvia de color per mostrar al jugador com de feble està el personatge.

- Si la vida és menor del 50% es posa de color groc.
- Si la vida és menor del 25% es posa de color vermell. A la figura 65 podem veure com es mostra.

Figura 65 HealthBar

6.5.2 BASIC ATTACK

L'atac bàsic s'ha creat com un objecte per si mateix per poder-lo registrar com a *prefab* i poder instanciar-lo en temps de joc a través de codi.

D'elements que el diferencïin als altres *GameObjects* físics com són els personatges és que aquest té un *collider* amb la casella "is trigger" activada, de forma que no xoca contra els altres objectes, sinó que els travessa. Una altra diferència es que el *RigidBody* està marcat com a "kinematic", es a dir que aquest no respon a les forçes o a les collisions, sino que el seu moviment està purament controlat des de codi. (Figura 66)

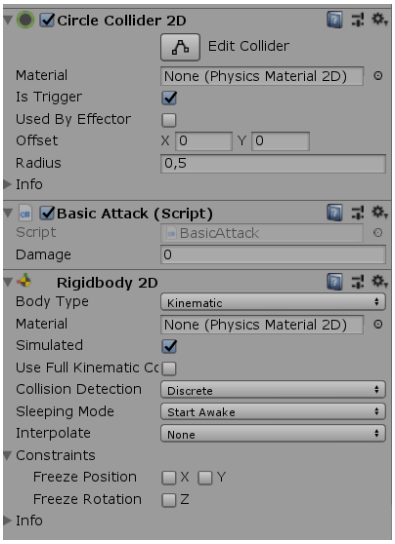


Figura 66 Basic attack

El *script* associat al "basic attack", només mou el projectil en la direcció que li han indicat i amb el rang del personatge que l'ha invocat, i mira si impacta algun personatge enemic. També li diu a la càmera que li enfoqui mentre viu.

6.5.3 GRENADE

La granada està formada per dos objectes per així poder tenir associats dos *colliders*. Un és el *collider* aplicat a la granada en si, el qual té un material personalitzat que fa que aquesta granada pugui rebotar. L'altre *collider* és el que fa referència al rang de l'explosió, el qual està desactivat fins que la granada explota.

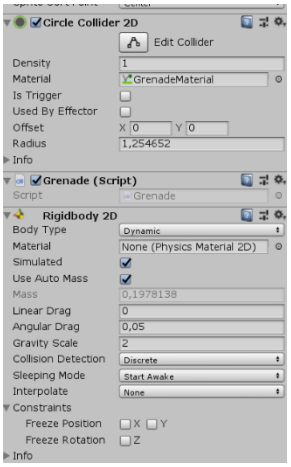


Figura 67 Grenade

Quan passa un cert temps des que la granada és instanciada, aquesta explota, atacant a tots els personatges propers per igual. També es crida a una funció d'una llibreria que s'ha descarregat de la tenda d'Unity de forma gratuïta que fa que la càmera tremoli, quedant un efecte molt immersiu. Aquest *script* al que es fa referència és *EZCameraShake.*

6.5.4 MENÚS

Els menús no tenen gaire misteri, són interfícies amb botons que carreguen escenes noves.

Al canviar d'escenes, s'ha afegit transicions per fer més estètic aquests canvis.

6.5.5 ANIMACIONS

Les animacions estan generades a partir de l'editor d'Unity. Primer es detallarà com s'ha creat l'animació i després com funciona l'animador d'Unity.

Per fer les animacions hi ha moltes maneres diferents, a Battleland s'ha fet cada part que s'ha d'animar en una capa separada per poder controlar-la millor i que no alteri la geometria de les altres parts. A la figura 68 es veu clarament.

Figura 68 Soldier en l'sprite editor

A continuació es col·loquen les parts en la posició original i es creen els ossos pels quals es regeix el personatge com es veu a la figura 69. S'ha de tenir especial compte en aquesta part perquè si es posen malament pots moure parts que no volies.

Figura 69 Esquelet del Soldier

Per acabar la part de preparar el personatge, hem d'indicar la influència que exerceixen els ossos. Així, quan moguem un os, la geometria al seu voltant reaccionarà a ell i es deformarà. (Figura 70)

Figura 70 Geometria Soldier

I ara si bé la part d'animar el personatge. Unity té un mòdul senzill d'animació que funciona com molts altres *softwares* d'animació, en el qual has d'indicar a cada instant de temps la posició de cada os del personatge. (Figura 71)

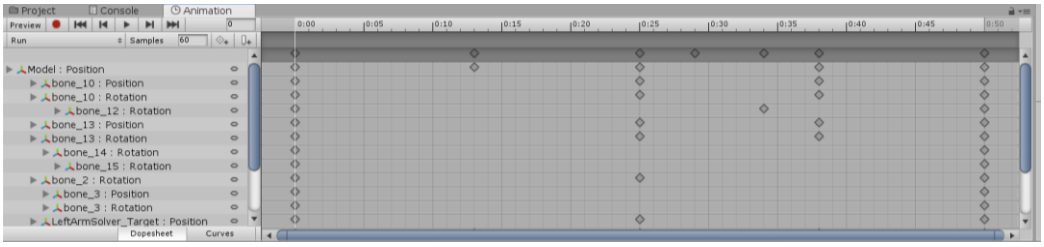
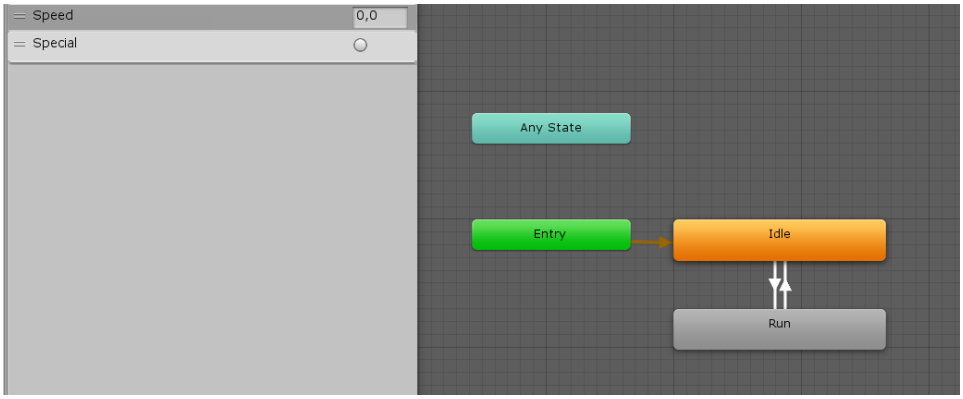


Figura 71 Animació Run Soldier

En el controlador d'animacions o animador, és on hi ha definides les transicions entre animacions. En la figura a continuació podem veure que hi ha una variable anomenada "speed" que es la que marca quina animació es mostra. Si la velocitat és més gran que 0, es posa l'animació de *run*, si per contra és igual a 0 es posa la de *Idle*.



Finalment mencionar que també s'ha utilitzat el *Package* gratuït "TextMesh Pro" descarregat des de la tenda Unity per fer molts dels textos del videojoc.

6.6 MULTIJUGADOR

A l'hora de fer el multijugador online, no es coneixia res sobre implementació Online, el qual ha sigut un error de no investigació que ha suposat un mal disseny de l'Online.

La idea era fer el joc local i després canviant alguna petita cosa fer la conversió a Online, però s'ha trobat amb moltes dificultats amb els algoritmes i l'estructura desenvolupada, pel qual hi ha hagut molta feina de reconvertir molt material que ja estava fet.

Per fer la implementació del joc Online, s'ha utilitzat "Unet", que es la forma que incorpora Unity de fer jocs multijugador en xarxa. El dolent, és que aquesta característica serà treta d'Unity d'aquí poc i ells mateixos et recomanen no utilitzar-la (Figura 72) perquè estan preparant una nova i millor. Això ha sigut una mica descoratjador perquè tot el treball invertit i el que s'ha après amb aquesta tecnologia servirà de poc o res en el futur.

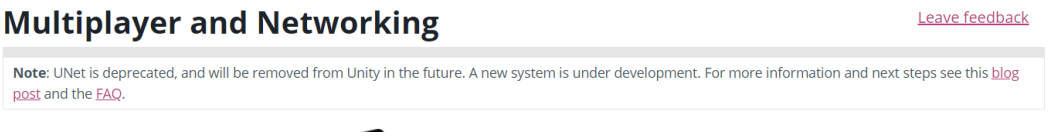


Figura 72 Avís d'Unity sobre Unet

Tot i els problemes comentats, s'ha desenvolupat d'una forma que s'ha intentat salvar el màxim possible els *scripts* generats i l'estructura del projecte. Com es veu a la figura següent, l'escena té una jerarquia similar al joc local.

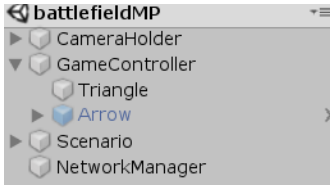


Figura 73 Estructura Joc Online

Les diferencies són que aquest cop no tenim cap equip instanciat dins de "GameController", ja que aquestes es generen quan un usuari es connecta i entra a l'escena. L'altre diferència és que ara tenim un nou objecte anomenat "NetworkManager", que és l'element que permet el joc en xarxa.

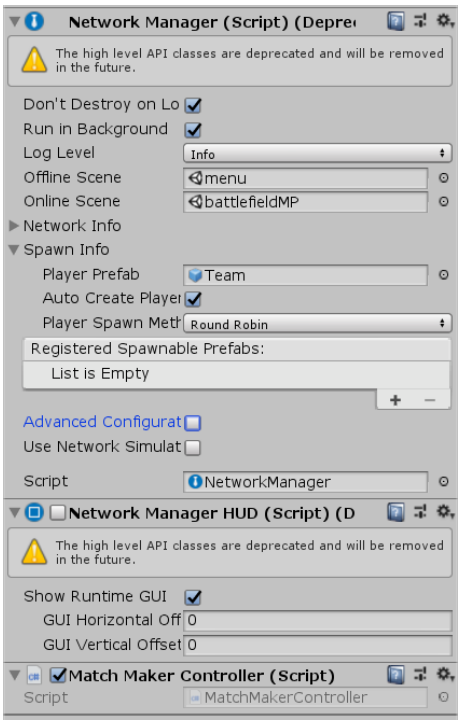


Figura 74 Network Manager

Com veiem a la figura superior, aquest objecte controla diferents aspectes del multijugador en xarxa, començant des de dalt cap a baix i pels aspectes més importants:

- Indica quina escena mostra si es perd la connexió, en aquest cas et redirigeix al menú principal.
- Indica l'escena que es carrega quan es connecta un jugador.
- Indica què es carrega quan es connecta un nou jugador, en aquest cas com és dit anteriorment es genera un *prefab* nou dissenyat per aquest mode que es diu "Team" i bàsicament és l'equivalent al "Team" que hi havia en el controlador de joc del mode local, amb la diferència que ara es genera des d'aquí i no està fet des de l'editor.
- S'indica la posició inicial on s'instancien els jugadors. Està seleccionat mètode "round robin" en comptes de "random", per poder controlar des del disseny la posició d'aparició de cada equip.
- En paràmetres avançats està marcat que només es permet un màxim de dues connexions.

Com veiem, també té un petit script anomenat "matchMakerController", aquest és un algoritme propi que serveix per fer un emparellament automàtic, sense haver de passar

per menús. El seu funcionament bàsicament és mirar si hi ha alguna partida creada per algun jugador, si és que si, connectar-me, si és que no, crear-ne una.
La idea és que el producte final també fos així per fer-ho àgil, però òbviament hi hauria un petit filtratge per fer que els jugadors competissin amb gent del seu nivell. Actualment no es mira res i et pot emparellar el millor jugador del món amb un que està començant.

Altres diferencies amb el que hi ha en el mode local és l'aparició de nous components necessaris pel joc en xarxa.
Primerament, el "GameController" de l'escena ara té un *script* anomenat "MPGameController" que deriva de "GameController". L'altre component nou és que ara aquest objecte té un "Network Identity", això crea una identitat única pel *GameObject* i utilitza aquesta identitat per a què el sistema en xarxa sigui conscient del *GameObject*. Per veure-ho amb claretat tenim la figura 75.

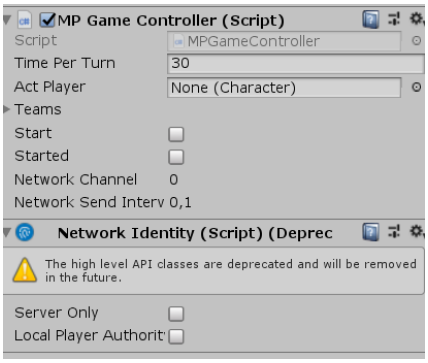


Figura 75 MPGameController

A continuació, s'ha creat un *prefab* nou a partir del que era un equip en el mode local i si han introduït alguns components com l'anteriorment mencionat *NetworkIdentity*, i també els elements *NetworkTransform* i *NetworkTransformChild*, els quals permeten que els *GameObjects* comparteixin la seva posició de servidor a client i de client a servidor, pel qual veurem el mateix moviment en les dues instancies de joc. A la figura de la pàgina següent (Figuta 76) podem veure aquests components.

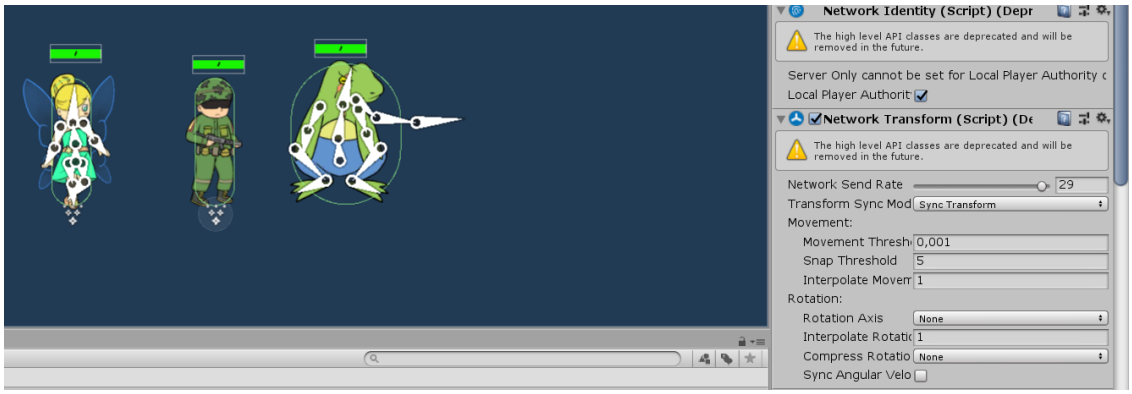


Figura 76 Team

Després, en el que recau al *gameplay*, s'ha d'entendre el funcionament d'*Unet*.

Primer tenim el *NetworkTransformChild*, el qual és el que fa que les posicions i transformacions en el món estiguin connectades entre instàncies. Després bàsicament es reprodueix el joc en forma local en les dues versions, però hi ha algunes coses que només seran executades en una instància (La que fa de servidor).

Per exemple, en Battleland la decisió de quin equip comença primer és executada només en servidor i després es comparteix la variable en qüestió per a que les dues instancies treballin en concordança. A continuació veiem que hem d'indicar que l'equip que comença és una variable sincronitzada i executar-la només en servidor. Llavors en la instància client la variable "startTeam" tindrà el mateix valor que al servidor.

```
[SyncVar]
private int startTeam;
```

```
if (isServer) startTeam = Random.Range(0, 2);
```

D'això hi ha bastants més exemples, com per exemple la vida, el temps, etc.

Per donar a entendre més el funcionament posarem com a exemple la vida. Si el client executa un atac i li pren vida a un personatge, el servidor no ho sabrà. Per això totes les accions importants s'executen en el servidor i en local només es fa una representació gràfica. Per això, quan el client executa un atac, li envia una comanda al servidor i l'atac és executat allà. Llavors, es marca que la variable vida és compartida i per tant el client canviarà el valor de la vida del personatge atacat.

Per lo restant, el funcionament d'una partida és igual.

Encara no s'ha acabat el desenvolupament d'aquesta part, per lo qual encara que l'emparellament de partides funciona correctament i els jugadors poden connectar-se a partides a través de la xarxa, la partida encara no es pot acabar.

6.7 RENDIMENT

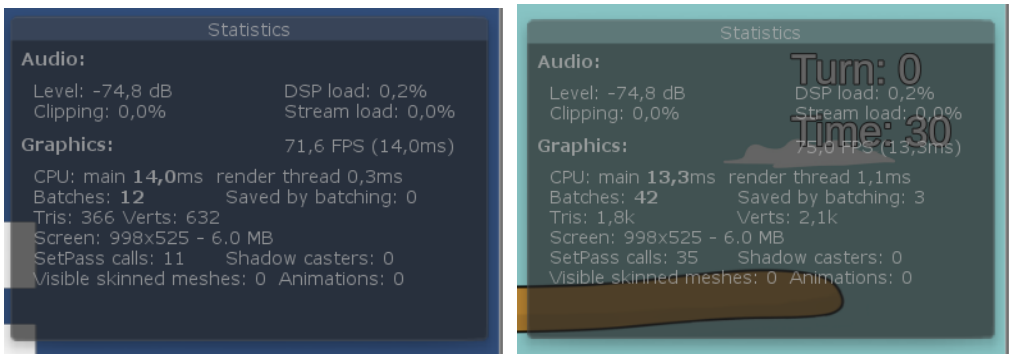


Figura 77 Estadístiques de rendiment

Com podem veure gràcies a les eines de mesura de rendiment que ofereix Unity (Figura 77), el joc es manté en uns FPS constants tota l'estona, tant en els menús com en el camp de batalla. Com és normal en el camp de batalla veiem que hi ha més treball a la part gràfica, com indica per exemple que hi ha més vèrtexs a renderitzar.

També el moviment dels personatges respon adequadament a l'input, sent imperceptible l'anomenat *lag input* (el retard des de que es prem un botó fins que el sistema el manega).

6.8 JOC CREUAT

Per implementar el multijugador creuat no s'ha de fer canvis en el codi, perquè Unet gestiona tot el relacionat amb aquest aspecte. Bàsicament, Unet ja va ser creat pensant el jocs multiplataforma, i el que fa és no diferenciar entre els usuaris en xarxa si són d'una plataforma o una altra. Per tant, l'únic que s'ha hagut de treballar en aquest aspecte és en fer un bon disseny del joc perquè la versió del joc no sigui diferent d'entre Windows i Android.

Pel que fa a les proves, s'ha comprovat que el joc funciona a la perfecció en ambdues plataformes, però que fins que no s'acabi de desenvolupar el multijugador en xarxa com es obvi, no es podrà dur a terme una partida creuada, tan sols establir una connexió.

7. RESULTATS

En aquest apartat mirarem els objectius que es van marcar en un principi i llavors farem valoració de si els hem assolit i amb quin grau de compliment.

Es recorda que l'objectiu principal era dissenyar un videojoc, amb la condició de simular un desenvolupament professional. L'altra condició més important que marcava el marc del projecte era que havia de ser multiplataforma per no excloure molta part de comunitat. Finalment es volia fer una versió prototip per mostrar el concepte en execució.

Per a tals qüestions, es va decidir marcar uns objectius específics:

- Explorar el mercat i definir un públic

Hi ha un apartat enterament dedicat a aquesta fita en la memòria, concretament el numero 2: estudi de viabilitat.

En mencionat apartat, es comença fent un estudi de mercat per saber a quina competència ens enfrontem, que proposen i com superar-los. Amb una cerca dels màxims exponents podem veure que ofereixen en els seus videojocs per llavors pensar que és el que el Battleland oferirà que sigui diferent i la gent aposti per ell.

Aleshores tenim la capacitat d'extreure quin és el públic objectiu amb la informació obtinguda gràcies a l'anàlisi de la competència. Sumat als coneixements adquirits durant el curs, s'extreu un públic objectiu molt específic i determinem el perfil de jugador.

- Analitzar la tecnologia

Amb aquest objectiu es fa referència a fer una anàlisi de com ha de ser el videojoc i quines eines tenim per poder desenvolupar-lo seguint la instrucció de què sigui multiplataforma.

Primerament, es va estudiar les dificultats de desenvolupar un videojoc per una plataforma i després fer la corresponent conversió en un altra plataforma. Es va veure que Unity permetia molt fàcilment exportar un tipus de projecte en un altre, i que les dificultats sorgien en el moment de dissenyar el videojoc, perquè s'ha de tenir en compte que ambdues plataformes tinguessin els mateixos tipus de controladors o una manera de simular els inputs que ho permetés.

Entre altres motius, es va decidir desenvolupar per Android i Windows perquè al analitzar les alternatives es va arribar a la conclusió que aquestes dues permetien un desenvolupament paral·lel i sense haver de canviar elements del joc per adequar-se a la tecnologia. Està explicat en l'apartat 5: Disseny.

- Definir el guió del joc

Bàsicament aquí es refereix a tot el relacionat amb dissenyar el videojoc, el qual està realitzat en l'apartat 5: Disseny.

Potser la part més essencial en un desenvolupament, és la que defineix com és el videojoc, com es veu, quines emocions transmet, etc.

- Definir l'abast del joc

Aquesta part està detingudament explicada durant l'apartat 3: Planificació.

Es defineix primerament un concepte del joc, i llavors es divideix per diferents requisits funcionals.

En aquesta secció també s'explica la metodologia a seguir i es detalla que forma part del videojoc i que s'ha decidit portar a terme pel prototip o demostració jugable.

- Dissenyar i implementar el joc local

Totes les explicacions per demostrar l'assoliment d'aquest objectiu es troben en els apartats 5 : Disseny i 6: Implementació i proves.

En el prototip l'usuari és capaç de jugar una partida local de principi a fi en un petit escenari i controlant tres personatges completament diferents, per el qual es considera assolit.

- Dissenyar i implementar el joc Online

Potser la part que ha sigut més complicada. Per canviar de joc local a joc online s'ha hagut de redissenyar alguns aspectes, pel qual es considera que no s'ha fet un bon disseny del videojoc en aquest aspecte.

S'hauria d'haver investigat més sobre el multijugador Online abans de començar a desenvolupar el joc local, i així es podria haver fet una estratègia que funciones per ambdós modes i només canviant alguna petita cosa, però no ha sigut el cas i davant la conversió a Online s'han trobat moltes dificultats inesperades.

De totes maneres s'ha aconseguit que la partida es reprodueixi en dispositius diferents,

tot i que a dia d'enquadernació d'aquesta memòria no es pot realment jugar una partida fins el final. S'espera que per a la presentació es pugui acabar i ensenyar una partida multijugador online.
L'explicació detallada sobre aquest objectiu es troba en els apartats 5: Disseny i 6: Implementació i proves.

- Fer-lo multiplataforma

Igual que es valora que el disseny del joc online no ha sigut satisfactori, aquí cal a dir que el disseny d'aquest joc per ser executat en plataformes diferents ha sigut idoni, ja que el joc és igualment de satisfactori tant en la versió d'Android com en Windows.
I això es demostra amb la versió prototip, amb la qual es pot veure que la interfície de botons virtuals és tot un encert per a un joc de mòbil i d'ordinador.
En les dues versions, tant en Android com en Windows, el joc s'executa a la perfecció, però com s'ha mencionat abans, el joc creuat deriva del multijugador Online i fins que aquest no funcioni a la perfecció, dos jugadors en diferents dispositius no podran gaudir de la veritable experiència del joc.

- Fer un correcte balanceig del joc

És un dels apartats més complicats en un videojoc d'aquestes característiques. El balanceig s'ha fet gràcies al disseny inicial dels personatges i després en jugar moltes partides s'ha pogut analitzar els punts fluixos i forts dels personatges per equilibrar-los. Per exemple, es va veure que l'habilitat del soldat, llançar una granada, era massa poc efectiva i es va decidir incrementar l'atac i el rang de l'explosió. Això s'ha realitzat molts cops abans de tenir la versió prototip, i segurament és una cosa que s'ha d'anar fent durant el manteniment i gràcies al *feedback* de la comunitat.

Pel que fa a si el videojoc compleix la legislació i normatives vigents, és un videojoc senzill pel qual no hi haurà cap problema. Entre d'altres, mai es guarda informació de caràcter personal, utilitza llibreries gratuïtes de lliure utilització, etc.
Té un apartat enterament dedicat al sistema PEGI dins la secció 5: Disseny.

8. CONCLUSIONS

Amb aquest treball he pogut posar en pràctica tot l'aprés durant el grau de disseny i desenvolupament de videojocs, i he pogut veure que encara hi ha moltes coses més per aprendre.

Battleland era una de les moltes idees que em varen sorgir a mesura que avançava cursos des que vaig començar el grau. Segurament s'hauria acabat portant a terme en algun moment de la meva vida, però gràcies al projecte he pogut aprofitar-ho per a realitzar aquest treball també. Amb el temps que s'ha disposat no s'ha pogut acabar de traslladar tota la imatge que tenia en el cap del videojoc, però ja serveix per començar a fer els meus primers videjocs i començar a emplenar el meu portafoli.

Tenia por que per culpa de convertir una idea pròpia de videojoc en el treball de final de grau fes que m'avorrís o no gaudis el desenvolupament, però crec que ha sigut una bona decisió perquè al fer una cosa que m'agrada s'encarava amb motivació i ganes, i també he pogut aprendre moltes coses que em continuaran servint a l'hora de desenvolupar nous jocs.

Tot i que ha sigut la part més dura per mi, fer aquest document també ha sigut important per veure com el disseny és molt important. Quan en alguna part m'emocionava de més i incorporava alguna cosa no planejada, veia que després m'apareixien conflictes molt difícils de tractar i que pràcticament havia de redissenyar alguna part o tornar enrere aquesta improvisació. No tinc dubtes que aquest treball em servirà de molt a l'hora d'encarar nous projectes.

S'ha tingut problemes sobretot amb el tema d'implementar el multijugador Online, perquè mai s'havia treballat aquest aspecte en Unity, però gràcies a algunes assignatures en les quals s'havia treballat el multijugador Online s'entenia com s'havia d'enfocar.

Lo millor ha sigut que mai m'he encallat gaire en cap tasca, gràcies a què hi ha molta informació a la xarxa sobre Unity i molta gent disposada a ajudar-te quan tens algun problema. La fase d'implementació per tant ha sigut bastant àgil.

Tot i que de moment només hi ha bones paraules, he de dir que per mi el prototip està lluny del que volia presentar. M'hauria agradat fer un parell de personatges més per a què tingués sentit la pantalla de formació i poder haver-la desenvolupat, i així demostrar

com de diferents poden ser les experiències d'una partida a una altra jugant amb un o un altre personatge.

No veig tant estrany que no hagi pogut arribar a les meves aspiracions, perquè no estic acostumat a projectes grans i en realitat no sabia exactament quan em portaria de temps cada cosa. A part, arribant al final em vaig adonar de què un projecte d'aquestes característiques ben executat se m'escapava de les dimensions temporals del treball de final de grau, sobretot perquè hi ha altres obligacions personals que impedeixen poder dedicar-li més temps al treball. Però així he pogut aprendre pel futur a fer planificacions que tinguin en compte més factors i siguin més realistes.

També, tot i que s'ha comentat que fer el document ha sigut una cosa bona per desenvolupaments futurs, crec que és una de les tasques que m'ha robat més temps, i molt d'aquest m'hauria agradat dedicar-lo a més desenvolupament i més feina artística.

Per tot el que s'ha dit, estic segur que pròximament continuaré amb el desenvolupament del videojoc per intentar realitzar tot el que falta del joc més el que es comenta en l'apartat de treball futur.

Per concloure, crec que s'ha complert amb els objectius del treball. En aquest treball volia fer el disseny d'un joc complet i intentar desenvolupar el màxim possible, establint prioritats de què era el més important com es declara al quadre d'autovaloració de l'apartat introductori. Amb el que menys content estic és amb la part artística, perquè m'agrada bastant aquest apartat i és el que més he deixat pel final, i no he pogut deixar-lo com volia per al prototip.

El disseny del joc conté tot el que en un principi s'ha pensat per Battleland, i el prototip té les parts per mi més importants i amb les quals es pot entendre la idea del joc. Es pot jugar una partida sencera i es pot veure que el protagonisme del joc el tenen els personatges amb les seves particularitats úniques que és el que es vol transmetre des del principi.

9. TREBALL FUTUR

En aquest apartat es parlarà sobre noves característiques o idees que es podrien incorporar al joc en un futur. Certament, penso que m'agradaria continuar endavant amb el videojoc una vegada finalitzat el projecte de final de grau, perquè hi ha moltíssimes idees de personatges i de funcionalitats que tenia pensades i que m'agradaria molt realitzar.

Nous personatges:

- **Francotirador**: tindria molt atac però poca vida, sumat a un rang d'atac dels més grans del joc i poca distància recorrible.
    - Habilitat: al prémer el botó d'habilitat, el següent atac bàsic serà un cop crític, infringint un 150% del dany base.
    - Passiva: quan més lluny estigui l'enemic, més dany fa l'atac bàsic.
- **Kamikaze**: personatge de característiques bastants generals tipus soldat, és a dir un personatge equilibrat.
    - Habilitat: immolar-se. Farà una explosió molt gran al voltant seu que afectarà tant enemics com aliats i tindrà un dany bastant alt.
    - Passiva: en morir, els altres personatges aliats incrementen el seu dany en 1 unitat.
- **Drac**: tindrà molta vida però poc dany i poc rang d'atac. La distància recurrible seria bastant elevada.
    - Habilitat: tira foc i deixa una zona amb llames, els personatges que estiguin a sobre de la zona perdran vida per cada torn que siguin allà.
    - Passiva: únic personatge del joc que podrà volar, permetent moure's en l'eix de les y.
- **Invocador**: personatge amb poca vida i poc atac, però bastant rang d'atac i poca distància recorrible.
    - Habilitat: invocar monstre. Invocarà una nova unitat i durant el torn podrà atacar amb ella, i després encara utilitzar el seu atac bàsic.
    - Passiva: quan hi ha un monstre invocat, se li augmenta l'atac i el rang d'atac bastant.

I així amb moltíssimes idees. És molt fàcil pensar personatges i molt divertit. Per el qual el joc tindria moltíssim contingut per mantenir-lo durant molt de temps.

Més modes:

- **Mode contra la màquina**: per quan no tens internet o algú amb qui jugar. Que també serviria per practicar noves formacions i estratègies abans d'entrar al camp de batalla.
    - Implementar diferents nivells de IA
- **Mode torneig Online**: mode en el qual fas una formació i has d'intentar guanyar un cert nombre d'adversaris sense canviar els membres de l'equip.
- **Mode cooperatiu i competitiu alhora**: Un mode molt diferent que fos només per partides sense puntuació en els quals les partides serien de 3 jugadors contra 3 jugadors, cadascun controlant un sol personatge. Podria ser molt divertit perquè faries estratègies amb els teus amics.

Més escenaris:

Com s'ha dit anteriorment en el disseny del joc, la idea és que amb cada incorporació de personatge, s'afegís un nou escenari que l'acompanyes en ambientació, tant en l'estil, com una mica que el personatge estigues en una situació confortable pel seu estil de joc.

Més objectes:

Introduir nous objectes en el *gameplay*, que es podrien introduir amb actualitzacions a mesura que es van tenint idees. Una de les idees dissenyades però que no s'ha pogut implementar pel prototip, perquè es varen establir altres objectius més prioritaris.

Sistema de missions o recompenses:

molts jocs ho utilitzen per a què el jugador tingui sempre coses a fer i accedeixi al joc almenys un cop per dia. Missions diàries o reptes que atorguessin petits premis diaris si es compleixen. Podrien ser missions molt senzilles com "utilitza un soldat en una partida competitiva" o "mata 3 enemics".

Resum

I acabar pràcticament tot el que s'ha dissenyat però se sortia de l'abast del treball de final de grau, com és implementar el sistema de rànquings, el guardat automàtic als núvols, una tenda per aconseguir personatges, etc. Que són coses que no s'han fet

perquè no té sentit, no tenim comunitat per pujar en el rànquing, ni dades que emmagatzemar, ni suficients personatges com per haver d'aconseguir-los en una hipotètica tenda. També introduir tot el relacionat amb la música i efectes de so.

I finalment, optimitzar-lo i poder publicar-lo en la Play Store i en alguna pàgina web per donar-lo a conèixer i aconseguir crear comunitat.

Conclusió

Bàsicament, com a treball futur acabar el disseny del joc sencer més seguir creant contingut com el que s'ha comentat de nous personatges i escenaris.

10.BIBLIOGRAFIA

1. Asociacion espanyola de videojuegos (AEVI) <<aevi.org>> 28/05/2018
http://www.aevi.org.es/web/wpcontent/uploads/2019/05/AEVI_Anuario_2018.pdf
2. Hektor profe <<hektorprofe.net >> 31/05/2015
https://docs.hektorprofe.net/escueladevideojuegos/articulos/fases-del-desarrollo-de-videojuegos/
3. Chris Tevez <<chrisestevez.me>> 25/08/2016
http://www.chrisestevez.me/2016/08/planificacion-videojuegos-solitarios.html
4. Unity3D <<unity3d.com>> 04/05/2018
https://unity3d.com/es/programming-in-unity
5. Brackeys <<youtube.com/Brackeys>>
https://www.youtube.com/channel/UCYbK_tjZ2OrIZFBvU6CCMiA
6. Sykoo <<youtube.com/Sykoo>>
https://www.youtube.com/channel/UCNJvwJ6daLmw4_gUKTw4cSg
7. Gamedev <<gamedev.stackexchange.com>> 14/09/2013
https://gamedev.stackexchange.com/questions/62174/matchmaking-system-in-unity
8. Gamedevelopertips << gamedevelopertips.com>> 21/03/2016
https://gamedevelopertips.com/unity-how-fade-between-scenes/
9. Assignatura 'Disseny Conceptual de videojocs' <<Universitat de Girona>>

11. ANNEXOS

Aquí es recopilen els diferents scripts realitzats:

Character.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.Networking;
using UnityEngine.UI;
using UnityStandardAssets.CrossPlatformInput;

public class Character : MonoBehaviour
{

    private float move = 0;
    private float distance = 0;
    private Slider distanceBarUI;
    private Slider healthBarUI;
    public int health;
    public int charDistance;
    public int damage;
    public int range;
    [HideInInspector]
    public int currHealth;
    [HideInInspector]
    public bool myTurn = false;
    public GameObject bullet;
    private TextMeshProUGUI characterTextUI;
    [HideInInspector]
    public int orientation;
    //[HideInInspector]
    public bool aiming = false;
    [HideInInspector]
    public int rotationZ = 0;
    public GameObject aimArrow;

    //[HideInInspector]
    public bool skill = false;
    //[HideInInspector]
    public bool attacked = false;
    [HideInInspector]
    public int kills = 0;

    public int turnsSpecial = 1;
    //[HideInInspector]
    public int turnsCharging = 0;

    // Start is called before the first frame update
    void Start()
    {
        kills = 0;
        currHealth = health;
        if (tag=="Team1") orientation = 1;
        else if (tag == "Team2") orientation = -1;
        distanceBarUI = GameObject.Find("DistanceBarUI").GetComponent<Slider>();
```

```
        healthBarUI = GameObject.Find("HealthBarUI").GetComponent<Slider>();
        characterTextUI =
GameObject.Find("CharacterTextUI").GetComponent<TextMeshProUGUI>();

    }

    // Update is called once per frame
    void Update()
    {

             Health();

        if (myTurn &&
GameObject.Find("CameraHolder").GetComponent<CameraScript>().target ==
gameObject)
        {
            UpdateUI();

            Move();

            Attack();


        }
    }


    void UpdateUI()
    {
        healthBarUI.value = currHealth;

healthBarUI.transform.Find("HealthText").GetComponent<TextMeshProUGUI>().text =
currHealth + "/" + health;
        distanceBarUI.maxValue = charDistance;
        distanceBarUI.value = Mathf.Abs(distance);
        characterTextUI.text = name;

        aimArrow.transform.position = new Vector3(transform.position.x +
orientation * 0.5f,transform.position.y + 2.1f, 0);


        if (orientation == -1) aimArrow.transform.eulerAngles = new Vector3(
            aimArrow.transform.eulerAngles.x, 180,
aimArrow.transform.eulerAngles.z);
        else if (orientation == 1) aimArrow.transform.eulerAngles =new Vector3(
            aimArrow.transform.eulerAngles.x, 0,
aimArrow.transform.eulerAngles.z);

    }

    void Attack()
    {
        if (aiming)
        {
            if (CrossPlatformInputManager.GetAxis("Vertical") > 0) rotationZ +=
2;
            else if (CrossPlatformInputManager.GetAxis("Vertical") < 0) rotationZ
-= 2;

            if (CrossPlatformInputManager.GetButtonDown("Cancel"))
            {
                aiming = false;
```

```
                aimArrow.SetActive(false);
            }

            else if (CrossPlatformInputManager.GetButtonDown("Basic") && attacked
== false && skill == false)
            {
                            attacked = true;
                            GameObject bulletClone = Instantiate(bullet, new
Vector3(transform.position.x + (orientation * 0.6f),
                    transform.position.y + 2.1f, 0),
aimArrow.transform.rotation);
                bulletClone.GetComponent<BasicAttack>().Set(damage, range, tag,
this);
                aimArrow.SetActive(false);


            }

            rotationZ = Mathf.Clamp(rotationZ, -70, 70);
            aimArrow.transform.eulerAngles = new
Vector3(aimArrow.transform.eulerAngles.x, aimArrow.transform.eulerAngles.y,
rotationZ);

        }

        if (CrossPlatformInputManager.GetButtonDown("Basic") && !aiming)
        {
            aiming = true;
            aimArrow.SetActive(true);
            transform.Find("Model").GetComponent<Animator>().SetFloat("Speed",
0);
        }

        else if (CrossPlatformInputManager.GetButtonDown("Special") && !aiming &&
turnsCharging>=turnsSpecial)
        {
                    skill = true;

             }
    }

    void Health()
    {
        GetComponentInChildren<Slider>().value = currHealth;

transform.Find("HealthBar").transform.Find("HealthText").GetComponent<TextMeshPro
UGUI>().text = currHealth + "/" + health;

    }

    void Move()
    {

             if (CrossPlatformInputManager.GetAxis("Horizontal") < 0 )
        {
            move = 0.2f;
            orientation = -1;
            transform.Find("Model").eulerAngles = new Vector3(0, 180);

        }

        else if (CrossPlatformInputManager.GetAxis("Horizontal") > 0)
```

```
        {
            move = 0.2f;
            transform.Find("Model").eulerAngles = new Vector3(0, 0);

            orientation = 1;
        }
        else move = 0;

        if (!aiming && distance + (orientation*move) > -charDistance && distance
+ (orientation*move) < charDistance)
        {
            transform.Find("Model").GetComponent<Animator>().SetFloat("Speed",
move);
            transform.Translate(orientation*move, 0, 0);
            distance += orientation*move;

        }

    }


    public void Reset()
    {
        myTurn = false;
        attacked = false;
        distance = 0;
        aiming = false;
        skill = false;
        transform.Find("Model").GetComponent<Animator>().SetFloat("Speed", 0);
        turnsCharging += 1;
        rotationZ = 0;
        aimArrow.transform.eulerAngles = new
Vector3(aimArrow.transform.eulerAngles.x, aimArrow.transform.eulerAngles.y,
rotationZ);
        SetOrderLayer("Default");
        transform.Find("HealthBar").GetComponent<Canvas>().sortingLayerName =
"Default";
             aimArrow.SetActive(false);

       transform.Find("Model").GetComponent<Animator>().ResetTrigger("Special");


       }

       public void SetAlternative()
       {
              transform.Find("Model").eulerAngles = new Vector3(0, 180);
              Component[] modelParts = GetComponentsInChildren<SpriteRenderer>();

              foreach (SpriteRenderer sprite in modelParts)
                     if (sprite.name != "Shadow")
                            sprite.color = Color.red;
       }

       private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.name == "DieZone")
        {
            attacked = true;

            Destroy(gameObject);
        }
```

```
    }

    public void SetOrderLayer(string layer)
    {
        Component[] modelParts = GetComponentsInChildren<SpriteRenderer>();

        foreach (SpriteRenderer sprite in modelParts)
            sprite.sortingLayerName = layer;
    }

    private void OnCollisionEnter2D(Collision2D collision) //provisional...
    {
        if (LayerMask.Equals(collision.gameObject.layer, gameObject.layer))
        {
            Physics2D.IgnoreCollision(collision.collider,
GetComponent<Collider2D>());
        }

    }

}
```

BasicAttack.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BasicAttack : MonoBehaviour
{
    private Vector2 inPos;
    private float speed = 0.4f;
    public int damage;
    private int range;
    Character character;


    // Start is called before the first frame update
    void Start()
    {
        inPos = transform.position;

GameObject.Find("CameraHolder").GetComponent<CameraScript>().Set(gameObject,
3.5f);
    }

    // Update is called once per frame
    void Update()
    {

        transform.Translate(speed, 0,0);
        if (Vector2.Distance(inPos, transform.position) > range)
Destroy(gameObject);
    }

    public void Set(int new_damage, int new_range, string new_team, Character
new_character)
    {
        damage = new_damage;
        range = new_range;
```

```
        tag = new_team;
        character = new_character;
    }


    private void OnTriggerEnter2D(Collider2D collision)
    {

        if (collision.gameObject.GetComponent<Character>() != null &&
collision.tag != tag)
        {
            collision.gameObject.GetComponent<Character>().currHealth -= damage;
            Destroy(gameObject);

            if (collision.gameObject.GetComponent<Character>().currHealth <= 0)
            {
                character.kills += 1;
                Destroy(collision.gameObject);
            }
        }
        else if (collision.gameObject.GetComponent<Character>() == null) {
            Destroy(gameObject);
        }
    }

}
```

ButtonControler.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class ButtonController : MonoBehaviour
{
    public Character actPlayer;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
          if (actPlayer == null) return;
        if (!actPlayer.skill && !actPlayer.aimArrow.active && actPlayer.myTurn )
        {
            transform.Find("RightButton").gameObject.SetActive(true);
            transform.Find("LeftButton").gameObject.SetActive(true);
            transform.Find("UpButton").gameObject.SetActive(false);
            transform.Find("DownButton").gameObject.SetActive(false);
            transform.Find("CancelButton").gameObject.SetActive(false);
            transform.Find("AttackButton").gameObject.SetActive(true);
            transform.Find("SpecialButton").gameObject.SetActive(true);
        }
        if (actPlayer.turnsCharging < actPlayer.turnsSpecial)
        {
```

```
            transform.Find("SpecialButton").GetComponent<Image>().color =
Color.gray;

transform.Find("SpecialButton").GetComponentInChildren<TextMeshPro>().text =
                "" + (actPlayer.turnsSpecial - actPlayer.turnsCharging);
        }
        else
        {
            transform.Find("SpecialButton").GetComponent<Image>().color =
Color.white;

transform.Find("SpecialButton").GetComponentInChildren<TextMeshPro>().text = "";
        }

             if (actPlayer.aimArrow.active)
             {
                    transform.Find("RightButton").gameObject.SetActive(false);
                    transform.Find("LeftButton").gameObject.SetActive(false);
                    transform.Find("UpButton").gameObject.SetActive(true);
                    transform.Find("DownButton").gameObject.SetActive(true);
                    transform.Find("CancelButton").gameObject.SetActive(true);
                    transform.Find("AttackButton").gameObject.SetActive(true);
                    transform.Find("SpecialButton").gameObject.SetActive(false);
             }
             if (actPlayer.skill)
             {
                    transform.Find("AttackButton").gameObject.SetActive(false);
                    transform.Find("SpecialButton").gameObject.SetActive(true);
                    transform.Find("RightButton").gameObject.SetActive(false);
                    transform.Find("LeftButton").gameObject.SetActive(false);
                    transform.Find("CancelButton").gameObject.SetActive(true);
             }


        if (GameObject.Find("CameraHolder").GetComponent<CameraScript>().freeCam
== true)
            transform.Find("CameraButton").GetComponent<Image>().color =

transform.Find("CameraButton").GetComponent<Button>().colors.pressedColor;
        else transform.Find("CameraButton").GetComponent<Image>().color =

transform.Find("CameraButton").GetComponent<Button>().colors.normalColor;
    }


}
```

CameraScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CameraScript : MonoBehaviour
{

    public GameObject target;
    public float offsetY;
    private float z = -10;
    private float timeWithoutTarget = 0.7f;
    public float minX, maxX;
    public float minY, maxY;
```

```
    public bool freeCam = false;
       public bool started = false;

    private float startOrthographicSize;

    // Start is called before the first frame update
    void Start()
    {
        startOrthographicSize =
GetComponentInChildren<Camera>().orthographicSize;
    }

    // Update is called once per frame
    void Update()
    {

              if (!started) return;

              if (!freeCam)
        {
            if (target != null)
            {
                float posX = Mathf.Clamp(target.transform.position.x, minX,
maxX);
                float posY = Mathf.Clamp(target.transform.position.y, minY,
maxY);
                transform.position = new Vector3(posX, posY, z);


            }

            else if (target == null && timeWithoutTarget != 0)
            {
                timeWithoutTarget -= Time.deltaTime;
                if (timeWithoutTarget < 0)
                {


GameObject.Find("GameController").GetComponent<GameController>().PlayNextTurn();

Set(GameObject.Find("GameController").GetComponent<GameController>().actPlayer.ga
meObject, 3.5f);
                    timeWithoutTarget = 0.7f;
                }
            }
        }
        else
        {
            float posX = Mathf.Clamp(transform.position.x, minX, maxX);
            float posY = Mathf.Clamp(transform.position.y, minY, maxY);
            transform.position = new Vector3(posX, posY, z);
        }


    }

    public void Set(GameObject new_target, float new_offsetY)
    {
        target = new_target;
        offsetY = new_offsetY;
    }
```

```
    public void ChangeMode()
    {
        freeCam = !freeCam;
        if (!freeCam)
        {
                    GetComponentInChildren<Camera>().orthographicSize =
startOrthographicSize;
        }
    }
}
```

Fairy.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

public class Fairy : MonoBehaviour
{
       private Character character;
       private Character teamMate;
       bool aux,aux2,aux3 = false;
       float miniTimer;
       Color teamMateColor;
       int vc = 0;

       // Start is called before the first frame update
       void Start()
    {
              character = GetComponent<Character>();
       }

    // Update is called once per frame
    void Update()
    {
              if (!character.attacked)
              {
                     vc = 0;
              }
              if (character.skill == true)
              {
                     if (CrossPlatformInputManager.GetButtonDown("Cancel"))
                     {
                            character.skill = false;

                     }
                     else if (CrossPlatformInputManager.GetButtonDown("Special")
&& aux3==false && character.attacked == false && character.skill == true &&
teamMate!=null)
                     {
                            aux3 = true;
                            teamMate.currHealth += 3;
                            if (teamMate.currHealth > teamMate.health)
                                   teamMate.currHealth = teamMate.health;
                            character.turnsCharging = -1;


                            Instantiate(GameObject.Find("Particles"),
teamMate.transform);
```

```
	GameObject.Find("Particles(Clone)").GetComponent<ParticleSystem>().Play();

                        Component[] modelParts =
teamMate.GetComponentsInChildren<SpriteRenderer>();

                        foreach (SpriteRenderer sprite in modelParts)
                                if (sprite.name != "Shadow")
                                        sprite.color = teamMateColor;
                        aux = true;

                }
                if (Input.touchCount > 0 || Input.GetMouseButtonDown(0))
                {

                        //Touch touch = Input.GetTouch(0);
                        //Ray ray =
Camera.main.ScreenPointToRay(touch.position);

                        Vector3 mousePos =
Camera.main.ScreenToWorldPoint(Input.mousePosition);
                        Vector2 mousePos2D = new Vector2(mousePos.x,
mousePos.y);

                        RaycastHit2D hit = Physics2D.Raycast(mousePos2D,
Vector2.zero);
                        if (hit.collider != null)
                        {
                                if (hit.collider.gameObject.tag == tag)
                                {
                                        teamMate =
hit.collider.gameObject.GetComponent<Character>();
                                        teamMateColor =
teamMate.transform.Find("Model").Find("cabeza").GetComponent<SpriteRenderer>().co
lor;
                                        Component[] modelParts =
teamMate.GetComponentsInChildren<SpriteRenderer>();

                                        foreach (SpriteRenderer sprite in
modelParts)
                                                if (sprite.name !="Shadow")
                                                        sprite.color =
Color.yellow;
                                }
                        }
                }
        }

        if (aux)
        {

                miniTimer += Time.deltaTime;
                if (miniTimer > 2)
                {
                        character.attacked = true;
                        aux = false;
                        miniTimer = 0;
                        Destroy(GameObject.Find("Particles(Clone)"));
                }
        }
```

```
            if (character.attacked && vc == 0 &&
GameObject.Find("CameraHolder").GetComponent<CameraScript>().target==null)
            {
                  StartCoroutine(Passive());
            }
            if (aux2)
            {

                  miniTimer += Time.deltaTime;
                  if (miniTimer > 3f)
                  {

      GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
null;
                        miniTimer = 0;
                        aux2 = false;

                  }
            }

      }
      IEnumerator Passive()
      {
            vc = 1;
            yield return new WaitForSeconds(0.6f);
            GameObject.Find("CameraHolder").GetComponent<CameraScript>().target
= transform.gameObject;
            transform.Find("Particles").GetComponent<ParticleSystem>().Play();
            character.currHealth += 1;
            if (character.currHealth > character.health)
                  character.currHealth = character.health;
            aux2 = true;
            aux3 = false;

      }


}
```

Frog.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

public class Frog : MonoBehaviour
{
      private Character character;
      // Start is called before the first frame update
      void Start()
    {
            character = GetComponent<Character>();
      }

      // Update is called once per frame
      void Update()
      {
            if (character.skill == true)
            {
                  if (CrossPlatformInputManager.GetButtonDown("Cancel"))
```

```
                        {
                                character.skill = false;

                        }
                        else if (CrossPlatformInputManager.GetButtonDown("Special")
&& character.attacked == false && character.skill == true)
                        {
                                character.attacked = true;

        transform.Find("Model").GetComponent<Animator>().SetTrigger("Special");
                                character.turnsCharging = -1;

                        }
                }

        }

        private void OnTriggerEnter2D(Collider2D collision)
        {

                if (collision.gameObject.tag!=character.tag &&
collision.gameObject.GetComponent<Character>()!=null
                        && collision.gameObject.GetComponent<FrogEffect>() == null)
                {
                        FrogEffect fe =
collision.gameObject.AddComponent<FrogEffect>();
                }



        }
}
```

FrogEffect.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityStandardAssets.CrossPlatformInput;

public class FrogEffect : MonoBehaviour
{
        GameObject ray;
        int turns = -1;
        bool aux = false;

        // Start is called before the first frame update
        void Start()
    {
                ray = Resources.Load<GameObject>("Ray");

                GetComponent<Character>().range /= 2;
                GetComponent<Character>().charDistance /= 2;
        }

    // Update is called once per frame
    void Update()
    {
                Debug.Log(turns);
                if (GetComponent<Character>().myTurn)
                {
```

```
                    if (CrossPlatformInputManager.GetButtonDown("Basic") ||
CrossPlatformInputManager.GetButtonDown("Special"))
                    {
                        int paralyzed = Random.Range(0, 2);

                        if (paralyzed == 1)
                        {
                            StartCoroutine(Paraliyed());
                        }
                    }

                    if (aux == false)
                    {
                        aux = true;
                        turns++;
                    }
                }
                else aux = false;

                if (turns == 2) Destroy(this);
            }

            IEnumerator Paraliyed()
            {
                GetComponent<Character>().myTurn = false;
                GetComponent<Character>().aimArrow.SetActive(false);
                GameObject instance = Instantiate(ray, transform);

                instance.transform.localPosition = new Vector3(0, 2.4f, 0);

                yield return new WaitForSeconds(2f);

                GetComponent<Character>().attacked = true;
                Destroy(instance);
            }

}
```

GameController.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;

public class GameController : MonoBehaviour
{
    private float turn;
    public float TimePerTurn = 30;
    private float timeLeft;
    private TextMeshProUGUI turnText;
    private TextMeshProUGUI timeText;
    private GameObject[] team1, team2;
    [HideInInspector]
    public Character actPlayer;
    private int countT1, countT2;
    private bool endGame;
    private int startTeam;
       int vc = 0;
```

```
    // Start is called before the first frame update
    void Start()
    {

        turn = 0;
        timeLeft = TimePerTurn;
              actPlayer = null;
        countT1 = countT2 = 0;

        team1 = new GameObject[3];
        for (int i = 0; i < 3; i++)
        {
            team1.SetValue(transform.Find("Team1").GetChild(i).gameObject, i);
            team1[i].tag = "Team1";
        }
        team2 = new GameObject[3];
        for (int i = 0; i < 3; i++)
        {
            team2.SetValue(transform.Find("Team2").GetChild(i).gameObject, i);
            team2[i].tag = "Team2";
        }

        turnText = GameObject.Find("TurnText").GetComponent<TextMeshProUGUI>();
        timeText = GameObject.Find("TimeText").GetComponent<TextMeshProUGUI>();
        endGame = false;

       GameObject.Find("CameraHolder").GetComponent<CameraScript>().started =
true;
        startTeam = Random.Range(0, 2);
        PlayNextTurn();
    }

    // Update is called once per frame
    void Update()
    {
        if (endGame = transform.Find("Team1").childCount == 0 ||
transform.Find("Team2").childCount == 0)
        {
            if (endGame = transform.Find("Team1").childCount == 0)
Debug.Log("Win: Team2");
            else Debug.Log("Win: Team1");
            Time.timeScale = 0;
        }

        turnText.text = "Turn: " + turn;
        timeText.text = "Time: " + Mathf.Round(timeLeft);
        timeLeft -= Time.deltaTime;

        if ((timeLeft < 0 || actPlayer == null || actPlayer.attacked) && vc==0)
        {
                    vc = 1;
                    //if (!actPlayer.attacked)
GameObject.Find("CameraHolder").GetComponent<CameraScript>().target = null;

       GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
null;
                    if (actPlayer != null)
            {
                if (actPlayer.tag == "Team1") countT1 = (countT1 + 1) % 3;
                else if (actPlayer.tag == "Team2") countT2 = (countT2 + 1) % 3;
            }
```

```
            //PlayNextTurn();

        }
    }



    Character ActPlayer()
    {
        Character player;

        if ((turn + startTeam) % 2 != 0) //team1
        {
            while (team1[countT1] == null) countT1 = (countT1 + 1) % 3;
            player = team1[countT1].GetComponent<Character>();
        }
        else //team2
        {
            while (team2[countT2] == null) countT2 = (countT2 + 1) % 3;
            player =  team2[countT2].GetComponent<Character>();
        }

        return player;
    }

    public void PlayNextTurn()
    {

        turn++;
              if (actPlayer!=null) actPlayer.Reset();
              vc = 0;
              actPlayer = ActPlayer();
        timeLeft = TimePerTurn;

        actPlayer.myTurn = true;

        actPlayer.SetOrderLayer("Character");

actPlayer.transform.Find("HealthBar").GetComponent<Canvas>().sortingLayerName =
"Character";
        GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
actPlayer.gameObject;

GameObject.Find("Triangle").GetComponent<FollowGameObject>().Set(actPlayer.gameOb
ject, 6f, 0);

GameObject.Find("ButtonsPanel").GetComponent<ButtonController>().actPlayer =
actPlayer;


    }
}
```

Grenade.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using EZCameraShake;
public class Grenade : MonoBehaviour
{
    private float timerExp;
```

```
    private float miniTimer = 0.3f;
    private int damage;
    private int power;
    private Character character;



    // Start is called before the first frame update


    void Start()
    {

GameObject.Find("CameraHolder").GetComponent<CameraScript>().Set(gameObject, 0);
        GetComponent<Rigidbody2D>().velocity =
transform.TransformDirection(Vector2.right * power);
        transform.Find("Explosion").GetComponent<CircleCollider2D>().enabled =
false;
        transform.Find("Explosion").GetComponent<SpriteRenderer>().enabled =
false;
    }
    public void Set(int new_damage, int new_power, float new_timerExp, Character
new_character)
    {
        damage = new_damage;
        power = new_power;
        timerExp = new_timerExp;
        character = new_character;

    }
    // Update is called once per frame
    void Update()
    {

        timerExp -= Time.deltaTime;
        if (timerExp < 0)
        {
            CameraShaker.Instance.ShakeOnce(4f, 4f, .1f, 1f);

                    GetComponent<Rigidbody2D>().velocity = new Vector2(0, 0);
            transform.Find("Explosion").GetComponent<CircleCollider2D>().enabled
= true;
            transform.Find("Explosion").GetComponent<SpriteRenderer>().enabled =
true;

            miniTimer -= Time.deltaTime;
            if (miniTimer < 0)
                Destroy(gameObject);
        }
    }


    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.GetComponent<Character>() != null)
        {
            collision.gameObject.GetComponent<Character>().currHealth -= damage;
            Destroy(gameObject);

            if (collision.gameObject.GetComponent<Character>().currHealth <= 0)
            {
                if (collision.tag != character.tag) character.kills += 1;
```

```
                Destroy(collision.gameObject);

            }
        }

    }

}
```

HealthBar.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HealthBar : MonoBehaviour
{

       float currHealth, health;
       bool focus = false;
       private void Start()
       {
              health = transform.parent.GetComponent<Character>().health;
              GetComponent<Slider>().maxValue = health;
       }

       private void Update()
       {

       }

       public void ChangeColor()
       {
              currHealth = GetComponent<Slider>().value;

              if (currHealth < health * 0.25)
              {
                     transform.Find("Fill
Area").Find("Fill").GetComponent<Image>().color = Color.red;
              } else if (currHealth < health * 0.5)
              {
                     transform.Find("Fill
Area").Find("Fill").GetComponent<Image>().color = Color.yellow;

              } else
              {
                     transform.Find("Fill
Area").Find("Fill").GetComponent<Image>().color = Color.green;
              }
       }

}
```

InitialScreen.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SceneManagement;
```

```
using UnityEngine.UI;

public class InitialScreen : MonoBehaviour
{
       TextMeshPro tapText;
       float timer = 1f;
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {


              if (Input.touchCount > 0 || Input.GetMouseButtonDown(0))
              {
                     Initiate.Fade("menu", Color.black, 2.0f);
              }

       }



}
```

Menu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class Menu : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {

    }

       public void ButtonLB()
       {
              {
                     Initiate.Fade("battlefield", Color.black, 2.0f);
                     //SceneManager.LoadScene("battlefield");
              }
       }
       public void ButtonExit()
       {
              {
                     Application.Quit();
              }
       }
```

```
	public void ButtonOB()
	{
		{
			Initiate.Fade("battlefieldMP", Color.black, 2.0f);
			//SceneManager.LoadScene("battlefieldMP");
		}
	}
}
```

MPGameController.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.Networking;

public class MPGameController : NetworkBehaviour
{

	private float turn;
	public float TimePerTurn = 30;
	//[SyncVar]
	private float timeLeft;
	private TextMeshProUGUI turnText;
	private TextMeshProUGUI timeText;
	private GameObject[] team1, team2;

	public Character actPlayer;
	private int countT1, countT2;
	private bool endGame;
	[SyncVar]
	private int startTeam;
	int vc = 0;
	public GameObject[] teams;
	public bool start = false, started = false;

	//[SyncVar]
	//public int idActPlayer;

	// Start is called before the first frame update
	void Start()
	{

		turn = 0;
		timeLeft = TimePerTurn;
		actPlayer = null;
		countT1 = countT2 = 0;

		team1 = new GameObject[3];

		team2 = new GameObject[3];


		turnText =
GameObject.Find("TurnText").GetComponent<TextMeshProUGUI>();
		timeText =
GameObject.Find("TimeText").GetComponent<TextMeshProUGUI>();
		endGame = false;
		if (isServer) startTeam = Random.Range(0, 2);

		turnText.text = "Turn: " + turn;
		timeText.text = "Time: " + Mathf.Round(timeLeft);
```

```
	}

	// Update is called once per frame
	void Update()
	{

		if (GameObject.FindGameObjectsWithTag("Team").Length == 2 && !start
&& !started)
		{
			start = true;

			if (start && !started)
			{
				teams = GameObject.FindGameObjectsWithTag("Team");
				teams[0].name = "Team1";
				teams[0].transform.parent =
GameObject.Find("GameController").transform;
				for (int i = 0; i < 3; i++)
				{

	team1.SetValue(teams[0].transform.GetChild(i).gameObject, i);

	Debug.Log(teams[0].transform.GetChild(i).GetComponent<Character>());

					team1[i].tag = "Team1";
				}
				//teams[0].GetComponent<TeamController>().SetEquip();

				teams[1].name = "Team2";
				teams[1].transform.parent =
GameObject.Find("GameController").transform;
				for (int i = 0; i < 3; i++)
				{

	team2.SetValue(teams[1].transform.GetChild(i).gameObject, i);

	teams[1].transform.GetChild(i).GetComponent<Character>().SetAlternative();
					team2[i].tag = "Team2";
				}
				//teams[1].GetComponent<TeamController>().SetEquip();
				PlayNextTurn();
				started = true;

	GameObject.Find("CameraHolder").GetComponent<CameraScript>().started =
true;

				GameObject.Find("SearchingImage").SetActive(false);

			}
		}

		if (started)
		{
			turnText.text = "Turn: " + turn;
			timeText.text = "Time: " + Mathf.Round(timeLeft);

			if (endGame = transform.Find("Team1").childCount == 0 ||
transform.Find("Team2").childCount == 0)
			{
				if (endGame = transform.Find("Team1").childCount == 0)
Debug.Log("Win: Team2");
				else Debug.Log("Win: Team1");
```

```
                        Time.timeScale = 0;
                    }
                    if ((timeLeft < 0 || actPlayer == null || actPlayer.attacked)
&& vc == 0)
                    {

                        vc = 1;
                        //if (!actPlayer.attacked)
GameObject.Find("CameraHolder").GetComponent<CameraScript>().target = null;

      GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
null;
                        if (actPlayer != null)
                        {
                            if (actPlayer.tag == "Team1") countT1 =
(countT1 + 1) % 3;
                            else if (actPlayer.tag == "Team2") countT2 =
(countT2 + 1) % 3;
                        }

                        PlayNextTurn();

                    }

                    timeLeft -= Time.deltaTime;


            }
      }


      Character ActPlayer()
      {
            Character player;

            if ((turn + startTeam) % 2 != 0) //team1
            {
                while (team1[countT1] == null) countT1 = (countT1 + 1) % 3;
                player = team1[countT1].GetComponent<Character>();

            }
            else //team2
            {
                while (team2[countT2] == null) countT2 = (countT2 + 1) % 3;
                player = team2[countT2].GetComponent<Character>();

            }

            //idActPlayer = player.GetInstanceID();
            return player;
      }



      public void PlayNextTurn()
   {

       turn++;
            if (actPlayer!=null) actPlayer.Reset();
            vc = 0;
            actPlayer = ActPlayer();
       timeLeft = TimePerTurn;
```

```
        actPlayer.myTurn = true;

        actPlayer.SetOrderLayer("Character");

actPlayer.transform.Find("HealthBar").GetComponent<Canvas>().sortingLayerName =
"Character";
        GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
actPlayer.gameObject;

GameObject.Find("Triangle").GetComponent<FollowGameObject>().Set(actPlayer.gameOb
ject, 6f, 0);

GameObject.Find("ButtonsPanel").GetComponent<ButtonController>().actPlayer =
actPlayer;


    }
}
```

PinchZoom.cs

```
using UnityEngine;

public class PinchZoom : MonoBehaviour
{
    public float orthoZoomSpeed;     // The rate of change of the orthographic
size in orthographic mode.
    public float speed;

    void Update()
    {
        if (GetComponentInParent<CameraScript>().freeCam==true)
        {
            if (Input.touchCount > 0 && Input.GetTouch(0).phase ==
TouchPhase.Moved)
            {
                Vector2 touchDeltaPosition = Input.GetTouch(0).deltaPosition;

                transform.parent.Translate(-touchDeltaPosition.x * speed *
Time.deltaTime, 0, 0); // y a 0 para pruebas
            }
        }

        // If there are two touches on the device...
        if (Input.touchCount == 2)
        {
            // Store both touches.
            Touch touchZero = Input.GetTouch(0);
            Touch touchOne = Input.GetTouch(1);

            // Find the position in the previous frame of each touch.
            Vector2 touchZeroPrevPos = touchZero.position -
touchZero.deltaPosition;
            Vector2 touchOnePrevPos = touchOne.position - touchOne.deltaPosition;

            // Find the magnitude of the vector (the distance) between the
touches in each frame.
            float prevTouchDeltaMag = (touchZeroPrevPos -
touchOnePrevPos).magnitude;
            float touchDeltaMag = (touchZero.position -
touchOne.position).magnitude;
```

```
            // Find the difference in the distances between each frame.
            float deltaMagnitudeDiff = prevTouchDeltaMag - touchDeltaMag;


            // ... change the orthographic size based on the change in distance
between the touches.
            GetComponent<Camera>().orthographicSize += deltaMagnitudeDiff *
orthoZoomSpeed;

            // Make sure the orthographic size never drops below zero.
            GetComponent<Camera>().orthographicSize =
Mathf.Max(GetComponent<Camera>().orthographicSize, 3.5f);
            GetComponent<Camera>().orthographicSize =
Mathf.Min(GetComponent<Camera>().orthographicSize, 30f);

        }
    }
}
```

Soldier.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;
using UnityStandardAssets.CrossPlatformInput;

public class Soldier : MonoBehaviour
{
    public GameObject grenade;
    private Character character;
    public int damage;
    public int power;
    public float timerExp;

       [SerializeField]
       private int buff;
       private int vc = 0;
       [SerializeField]
       private float miniTimer;
       private bool aux = false;


       private void OnEnable()
    {
        character = GetComponent<Character>();

    }

    // Update is called once per frame
    void Update()
    {
        if (character.skill == true)
             {
                    character.aiming = true;
                    character.aimArrow.SetActive(true);

                    if (CrossPlatformInputManager.GetButtonDown("Cancel"))
                    {
                           character.aiming = false;
                           character.aimArrow.SetActive(false);
```

```
                        character.skill = false;


                }

                else if (CrossPlatformInputManager.GetButtonDown("Special")
&& character.attacked == false && character.skill == true)
                {
                        GameObject grenadeClone = Instantiate(grenade, new
Vector3(transform.position.x + (character.orientation * 0.6f),
                                transform.position.y + 2.1f, 0),
character.aimArrow.transform.rotation);
                        grenadeClone.GetComponent<Grenade>().Set(damage,
power, timerExp, character);
                        character.aimArrow.SetActive(false);
                        character.attacked = true;
                        character.turnsCharging = -1;


                }
        }

        if (GetComponent<Character>().kills == 1 && vc == 0)
        {
                aux = true;

    GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
gameObject;

    transform.Find("HealthBar").transform.Find("Buffs").GetComponent<Image>().
enabled = true;

    transform.Find("HealthBar").transform.Find("Buffs").GetComponentInChildren
<TextMeshPro>().text = "+2";
                GetComponent<ParticleSystem>().Play();
                vc += 1;
                GetComponent<Character>().damage += buff;
        }
        else if (GetComponent<Character>().kills == 2 && vc == 1)
        {
                aux = true;

    GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
gameObject;

    transform.Find("HealthBar").transform.Find("Buffs").GetComponentInChildren
<TextMeshPro>().text = "+4";
                GetComponent<ParticleSystem>().Play();
                vc += 1;
                GetComponent<Character>().damage += buff;
        }

        if (aux)
        {

                miniTimer -= Time.deltaTime;
                if (miniTimer < 0)
                {

    GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
null;
                        aux = false;
```

```
                        miniTimer = 2;

                    }
                }

        }


}
```

SoldierPassive.cs

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.UI;

public class SoldierPassive : MonoBehaviour
{
    [SerializeField]
    private int buff;
    private int vc = 0;
    [SerializeField]
    private float miniTimer;
    private bool aux = false;
    // Start is called before the first frame update
    void Start()
    {
        vc = 0;
    }

    // Update is called once per frame
    void Update()
    {
        if (GetComponent<Character>().kills == 1 && vc == 0)
        {
            aux = true;
            GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
gameObject;

transform.Find("HealthBar").transform.Find("Buffs").GetComponent<Image>().enabled
= true;

transform.Find("HealthBar").transform.Find("Buffs").GetComponentInChildren<TextMe
shPro>().text = "+2";
            GetComponent<ParticleSystem>().Play();
            vc += 1;
            GetComponent<Character>().damage += buff;
        }
        else if (GetComponent<Character>().kills == 2 && vc == 1)
        {
            aux = true;

       GameObject.Find("CameraHolder").GetComponent<CameraScript>().target =
gameObject;

       transform.Find("HealthBar").transform.Find("Buffs").GetComponentInChildren
<TextMeshPro>().text = "+4";
            GetComponent<ParticleSystem>().Play();
            vc += 1;
            GetComponent<Character>().damage += buff;
        }
```

```
        if (aux)
        {

            miniTimer -= Time.deltaTime;
            if (miniTimer < 0)
            {

GameObject.Find("CameraHolder").GetComponent<CameraScript>().target = null;
                aux = false;
                miniTimer = 2;

            }
        }

    }

}
```

12. MANUAL D'INSTAL·LACIÓ I D'USUARI

12.1 MANUAL D'INSTAL·LACIÓ

En aquest capítol s'explica com instal·lar el videojoc Batteland sigui en la plataforma d'Android com sigui en un ordinador amb Windows.

Android

1. El primer és descarregar l'arxiu executable Battleland.apk (Figura 78) en el dispositiu i des de el dispositiu mòbil fer clic.

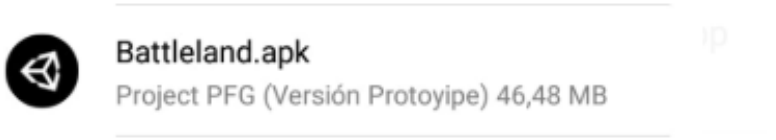


Figura 78 fitxer apk

2. Després de fer clic se'ns obrirà una nova pantalla (Figura 79) preguntant-nos si volem instal·lar l'aplicació, cliquem sobre el botó "Instalar".

Figura 79 Instal·lació de Battleland

3. En quant cliquem a instal·lar, apareixerà una barra de càrrega. És possible que aparegui la següent pantalla (Figura 80). Això és a causa de que com bé posa en el missatge que no es reconeix la desenvolupadora. Instal·lat de totes formes.

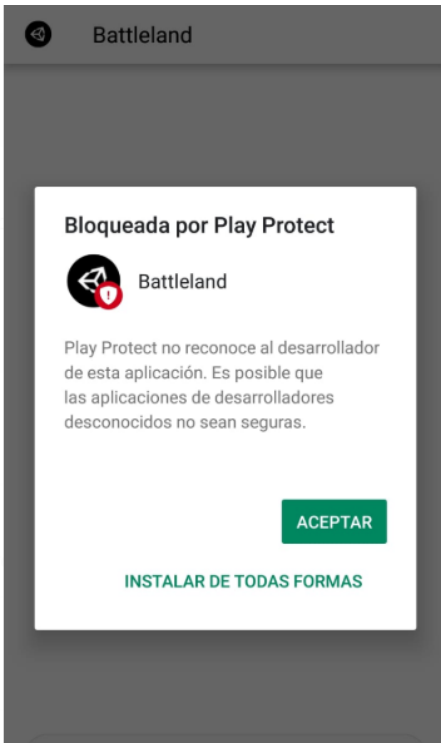


Figura 80 Bloqueig de Play Protect

4. Llavors arribarem a la pantalla que indica que l'aplicació ha sigut instal·lada amb èxit (Figura 81). Podem clicar a "Listo" per sortir de la instal·lació o a "Abrir" per entrar directament al joc.

Figura 81 aplicació instal·lada en el dispositiu

Windows

1. Per poder jugar a Battleland en la seva versió d'ordinador, primer necessitem descarregar-lo en el nostre equip (Figura 82).

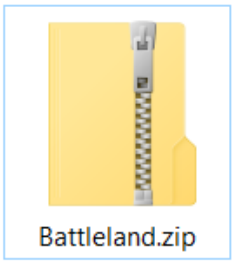


Figura 82 Carpeta contenedora de Battleland

2. Aquest ve comprimit en un fitxer zip per a què ocupi menys espai i així optimitzar la seva descarrega. Per descomprimir-lo, hem de fer clic dret i buscar l'opció de descomprimir. Hi ha diverses opcions per extreure el contingut, es recomana "Extraer" en "Battleland\" per així tenir-ho en una carpeta contenidora com es fa a la figura següent.

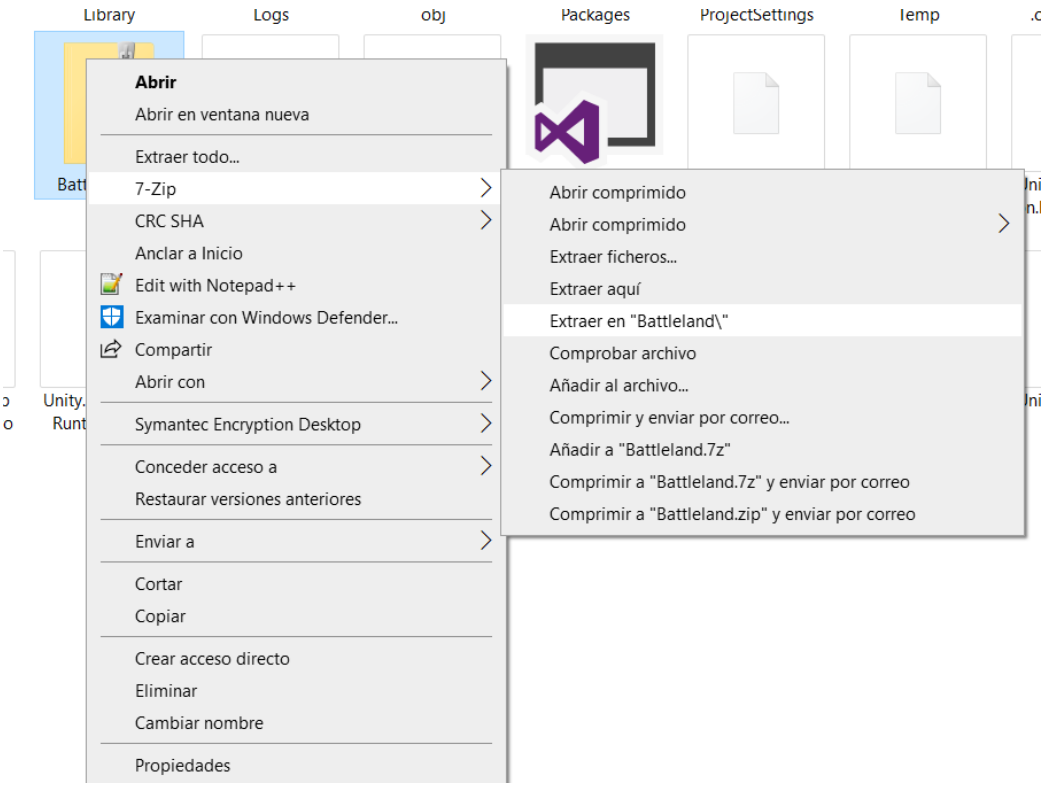


Figura 83 Extracció de Battleland

3. Després podem entrar a la carpeta extreta i allà podem veure l'executable del nostre joc (Figura 84). Al fer doble clic se'ns obrirà el joc.

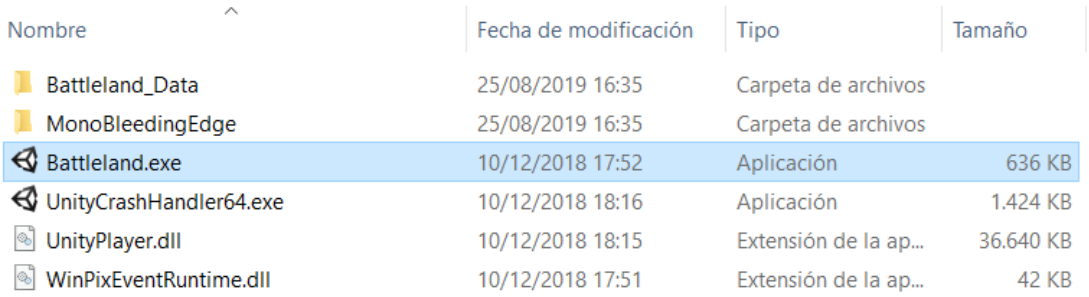


Figura 84 Executable de Battleland

11.2 MANUAL D'USUARI

A continuació, s'adjunta en forma de capítol un petit manual d'usuari mostrant el funcionament de l'aplicació prototip i detallant les accions que es poden dur a terme.

Primer, en executar el videojoc, apareixerà una pantalla amb el logotip d'Unity (*splash screen*), la qual és obligatòria si empres el motor en la seva versió gratuïta (Figura 85).

Figura 85 Splash screen d'Unity

Després, ja entrem en el què és la pantalla inicial del videojoc Battleland (Figura 86). Aquesta pantalla es tan sols de decoració. Romandrà així mentre el jugador no faci clic o toqui la pantalla.

Figura 86 Pantalla inicial

Llavors tres esperar un petit temps ens portarà a la pantalla de selecció de mode (Figura 87) . Aquí podem seleccionar “local battle”, “online battle” o “Exit”. Al final pel prototip no s’ha fet les opcions i s’ha canviat per el botó de Sortir.

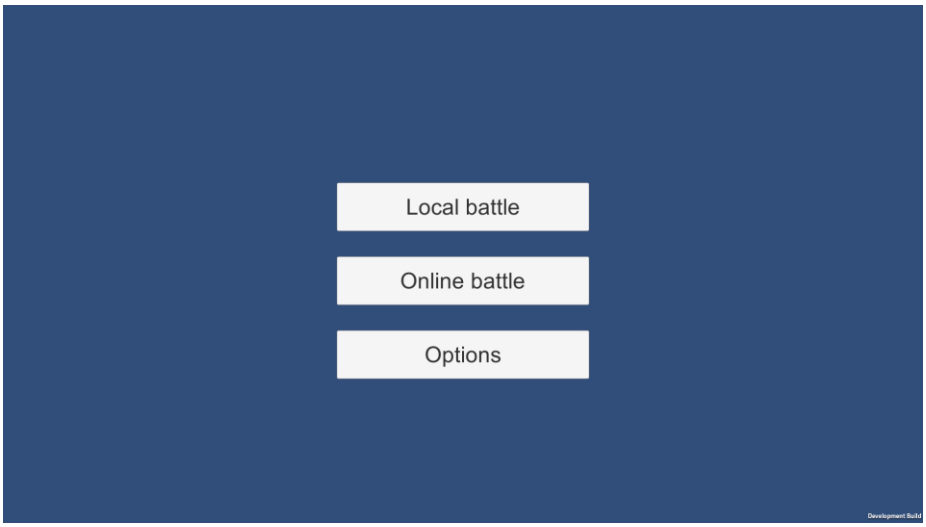


Figura 87 Menú

Si es clica al mode de “local battle”, instantàniament ens portarà al camp de batalla (Figura 88).

Aquest mode està pensat per a què el juguin dues persones des del mateix dispositiu, en el que s’anomena multijugador *hotseat*.

El controlador de joc decideix a l’atzar quin dels dos equips comença L’objectiu del joc és eliminar les unitats de l’equip contrari. Per diferenciar els equips, l’equip 2 tindrà un color alternatiu.

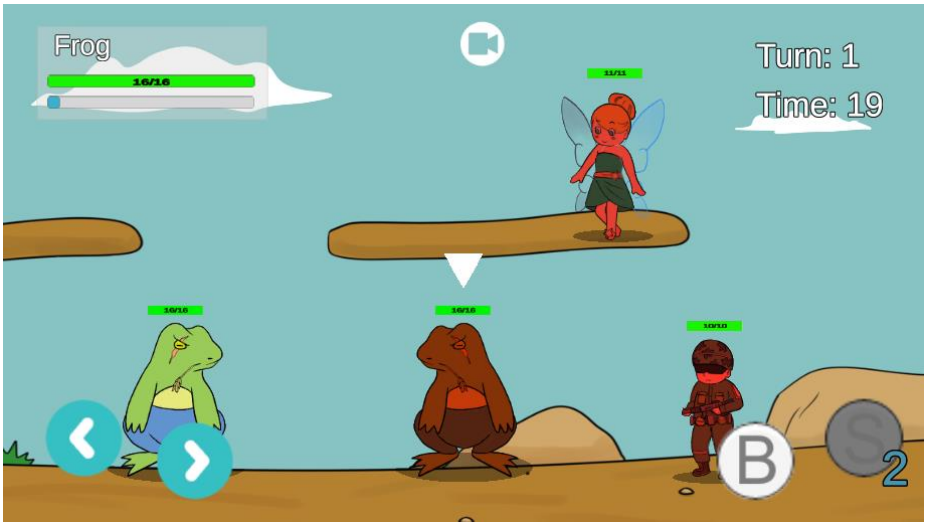


Figura 88 Camp de Batalla

Al personatge que li toqui moure's li apareix una fletxa indicadora a sobre. En la part superior podem veure informació de la partida, a l'esquerra algunes característiques del personatge actual i a la dreta el torn en el qual es troba la partida i el temps que queda per finalitzar el torn actual. En el teu torn pots:

- Moure't amb els botons de la cantonada inferior esquerra. El personatge es pot moure un determinat nombre d'unitats, depenent de les seves característiques.
- Atacar prement el botó de la cantonada inferior dreta. En seleccionar aquest mode, el jugador passa a estar apuntant. En tornar a clicar el boto, disparará un projectil en la direcció a la qual apunti el personatge.

Si han passat suficients torns, el personatge podrà utilitzar la seva habilitat especial

- En prémer el botó d'habilitat especial, apareixerà un nou tipus d'acció. Aquesta canvia depenent del personatge.
    - **Soldier**: passarà al mode d'apuntat, que té la mateixa mecànica que el de l'atac bàsic. Una vegada hem decidit on disparar, al tornar a prémer el botó, llançarà una granada que provoca dany en àrea (també afecta unitats aliades!).
    - **Fairy**: Podrà seleccionar una unitat del seu equip (ella inclosa) i en tenir-la seleccionada i tornar a prémer el botó d'habilitat, la curarà 3 unitats de vida.
    - **Frog**: traurà la llengua i totes les unitats que siguin colpejades se'ls hi aplicarà l'estat alterat "frog". En aquest estat i durant dos torns, el personatge té un 50% de probabilitats de quedar paralitzat i no poder atacar, se li redueix en un 50 % la distància recurrible i un altre 50 % en el rang de l'atac.

A mesura que avança la partida, si succeeixen certs esdeveniments o es compleixen certes condicions, són activades les passives dels personatges:

- **Soldier**: en matar una unitat enemiga, se li incrementa la força en 2 unitats.
- **Fairy**: en acabar el seu torn, es cura sempre 1 unitat de vida.
- **Frog**: cada dos cops que un personatge amb l'estat "frog" es paralitzat, se li augmenta en 1 els torns necessaris per utilitzar l'habilitat (contra-passiva)

El *gameplay* del mode multijugador Online és exactament el mateix que el descrit al mode local *hotseat*, l'únic que aquesta vegada es juga contra un jugador que està en un dispositiu diferent.

Finalment podem sortir del joc prement el botó "Exit".