UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Desenvolupament d'un asset de Generació de FX Pixel Art

Treball Final de Grau

Grau en Disseny i Desenvolupament de Videojocs

Cognoms: Trueba López Nom: Ian

Pla: 2014

Director: Díaz Acedo, Iñaki

Índex

Resum

L'objectiu d'aquest projecte és realitzar un *asset* pel motor de jocs Unity el qual permeti crear efectes especials en estil *pixel art* de manera automàtica. Per aconseguir-ho s'ha desenvolupat una interfície intuïtiva que permeti a qualsevol tipus d'usuari personalitzar el seu sistema de partícules i exportar-lo tant per projectes dins del motor com en format imatge per fer-lo servir en d'altres aplicacions.

Al llarg del document s'expliquen diferents tècniques de *pixel art* i el seu desenvolupament amb l'ús de *shaders* i la llibreria d'editor de Unity. El resultat del projecte serà llançat a la Unity Asset Store, explicant també en aquest treball els passos a seguir per realitzar una publicació en aquesta plataforma.

Paraules Clau

FX, partícules, pixel art, shader, UI, Unity, videojoc.

Enllaços

- GitHub: https://github.com/ianexe/PixelParticleEditor
- Unity Asset Store: https://assetstore.unity.com/packages/slug/148182
  (Enllaç subjecte a l'aprovació del projecte a l'Asset Store)

Índex de taules

Índex de figures

Glossari

Pixel Art: Representació gràfica basada en píxels que es va popularitzar a inicis de la indústria del videojoc.

FX: Abreviatura que fa referencia als efectes especials.

Indie: Equips de desenvolupament petits, independents a grans empreses.

Asset: Element que forma part del desenvolupament d'un joc.

Pixel Perfect: Concepte que es podria descriure com a la representació més fidel possible al píxel.

Shading: Conjunt de tècniques d'ombrejat.

Hue Shifting: Modificació d'una paleta de color amb l'objectiu d'aportar més contrast als colors que la formen.

Dithering: Degradat de colors fet a partir de diferents patrons a gust de l'artista, sense fer servir cap color intermedi.

Antialiasing: Degradat de colors fet a partir de l'addició de colors intermedis en el punt de contrast.

Shader: Programa que determina la manera com es pinta un element en pantalla.

UI: Abreviació de "User Interface", és a dir, interfície d'usuari.

MVP: De l'anglès Minimum Viable Product. Producte en fase de producció que compta amb funcionalitats bàsiques per poder ser utilitzat

Beta: Fase de desenvolupament de software on ja es compta amb totes les funcionalitats integrades.

Back-end: part interna del desenvolupament, on es programen les funcions per les quals el programa ha estat creat.

Front-end: part desenvolupament on es programaran els elements que es mostraran en pantalla amb els quals l'usuari interaccionarà.

Int: atribut de programació, el qual representa un valor numèric sense decimals.

Float: atribut de programació, el qual representa un valor numèric amb decimals.

Bool: atribut de programació, el qual representa un valor binari.

Top Down: Vista 2D en la que l'angle de la càmera està lleugerament inclinada en l'eix Y però no hi ha possibilitat d'apreciar l'eix Z.

Skeumorfisme: Corrent de disseny popularitzada per l'empresa Apple en la qual es mostren elements virtuals simulant les seves contraparts reals.

Slider: element de UI format per una barra i un botó el qual es pot lliscar per tot el recorregut d'aquesta barra. La posició del botó determina un valor definit pel desenvolupador.

1. Introducció

1.1 Motivació

Des de ben petit havia volgut formar part de la indústria del videojoc, i va ser el *pixel art*[1] el que em va obrir les portes a aquest mon. El que era un hobby s'ha anat convertint poc a poc en una activitat professional, arribant a fer encàrrecs per a diferents particulars i petits estudis.

Per una altra banda, al començar el grau de Disseny i Desenvolupament de Videojocs la programació va cridar molt el meu interès, fent que aparegués un dels grans dubtes de cara al meu futur professional: Quin és el perfil que millor s'adapta a mi? D'una banda m'encanta la programació, però la meva passió és, des de fa molts anys, el *pixel art*.

És per aquesta raó que he decidit fer un TFG que combini ambdues disciplines, fent una eina que m'ajudi a millorar en els dos àmbits. Els FX[2] són un dels aspectes més importants a l'hora de donar feedback al jugador, i com a artista m'interessa molt investigar quina seria la manera més òptima per crear-los, a més de ser un bon punt de partida per posar a prova els meus coneixements tant de *pixel art* com de programació.

1.2 Formulació del problema

Un dels grans reptes que suposa crear un projecte en *pixel art* és la limitació en aspectes com les paletes de color utilitzades, les dimensions dels elements o les animacions que aquests presenten. A l'art tradicional hi ha un munt de solucions automatitzades per aquests problemes, però al *pixel art* és molt important respectar el màxim possible l'estil, el qual dificulta l'aplicació d'aquestes solucions obligant a l'artista a retocar els dibuixos manualment.

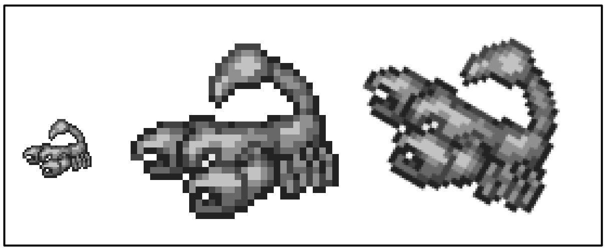


Figura 1.1: *Escorpí Pixel Art amb rotació automàtica*

[1] El Pixel Art és la representació gràfica basada en píxels que es va popularitzar a inicis de la indústria del videojoc. A dia d'avui aquest estil artístic s'utilitza molt sovint en desenvolupaments "Indies", és a dir, els realitzats per equips petits independents a grans empreses.
[2] El mot "FX" és una abreviatura que fa referència als efectes especials. En aquest projecte els efectes amb els que es treballaran són efectes especials visuals per videojoc.

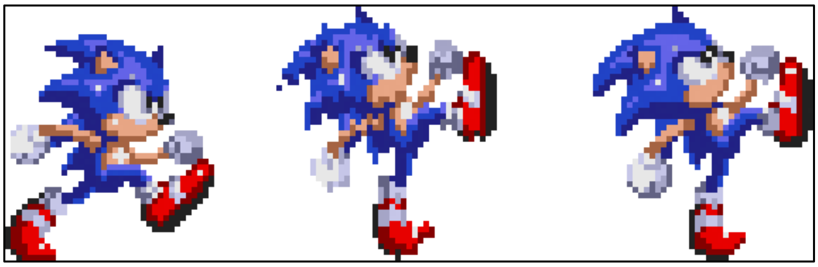


Figura 1.2: *Sonic Pixel Art amb rotació automàtica i retoc manual*

A més, el *pixel art* és conegut per ser utilitzat majoritàriament per equips petits i estudis indie, els quals no acostumen a tenir una gran font de recursos. Per això, sempre resulta beneficiós l'ús d'eines que ajudin a produir material de manera més ràpida intentant sacrificar el mínim de qualitat possible. Pel que respecta a FX hi ha molta varietat de solucions a aquest problema, però l'oferta quant a efectes especials *pixel art* és molt reduïda.

Analitzant les eines disponibles al mercat s'ha trobat "Pixel FX Designer", aplicació que soluciona el problema de la generació de FX *pixel art*. Mentre que aquest programa és totalment independent, resultaria interessant trobar una eina d'aquestes característiques integrada dins d'algun motor de joc.

El motor més popular en el desenvolupament de jocs 2D és Unity, el qual és utilitzat per un gran número d'estudis independents. Tenint en compte aquest factor, el desenvolupament d'aquest projecte es centrarà en aquest motor de joc, creant un *asset*[3] totalment integrat que pugui generar partícules i efectes especials en *pixel art* de manera automàtica.

1.3 Objectius generals del TFG

Aquest TFG té com a objectiu desenvolupar una eina que permeti crear efectes especials en estil *pixel art* de manera automàtica. El desenvolupament es farà amb Unity i una vegada finalitzat el projecte es pujarà a l'Asset Store d'aquest motor.

Per satisfer les necessitats dels usuaris familiartizats amb el *pixel art* s'afegiran diferents opcions de limitació de paletes de color, opcions d'animació i exportació del efectes per acabar de polir-los amb eines de dibuix manual.

[3] En la industria del videojoc es coneix com a "asset" a un element que forma part del desenvolupament d'un joc. Pot variar des de models 3D fins a eines d'intel·ligència artificial. En aquest projecte es desenvoluparà un asset enfocat a la generació automàtica d'efectes especials.

1.4 Objectius específics del TFG

El desenvolupament d'aquest projecte va destinat a la creació d'efectes especials *pixel art* tant per a usuaris experts com per a poc experimentats en l'apartat artístic.

Al voler satisfer ambdós perfils, d'una banda es volen obtenir FX fidels als estàndards del *pixel art*, per així reduir la quantitat de retocs necessaris pels usuaris avançats. Per l'altra banda es buscarà una fàcil experiència pels usuaris novells sense haver de sacrificar la qualitat del producte final.

1.5 Abast del projecte

En acabar el projecte es tindrà una eina per Unity que comptarà amb les següents característiques:

- Creació de partícules a partir de formes bàsiques
- Opcions de limitació de paletes
- Opcions de moviment i animació
- Exportació de partícules en format *sprite sheet* .png i .gif

El *target* al que va dirigit aquest producte seria qualsevol desenvolupador de Unity que tingui intenció de realitzar un joc en *pixel art* o, al menys, necessiti implementar efectes especials en aquest estil. Tant usuaris nous com experts es podran beneficiar de les opcions oferides per aquest *asset*.

De totes maneres, també pot arribar a ser utilitzat per desenvolupadors que treballin amb altres motors, gràcies a les opcions d'exportació implementades, a més d'artistes que vulguin afegir FX a les seves peces en molt poc temps.

2. Estat de l'art

2.1 Pixel Art

El *pixel art* va ser la forma de representació gràfica que va caracteritzar els videojocs des del seu origen fins l'arribada de tecnologies que van permetre renderitzar polígons tridimensionals i imatges d'alta qualitat. Mentre que a mitjans dels anys 2000 semblava que aquest estil artístic estava destinat a desaparèixer, l'aparició del sector *indie* i jocs com Hyper Light Drifter o Shovel Knight pocs anys després va suposar el renaixement del *pixel art*.

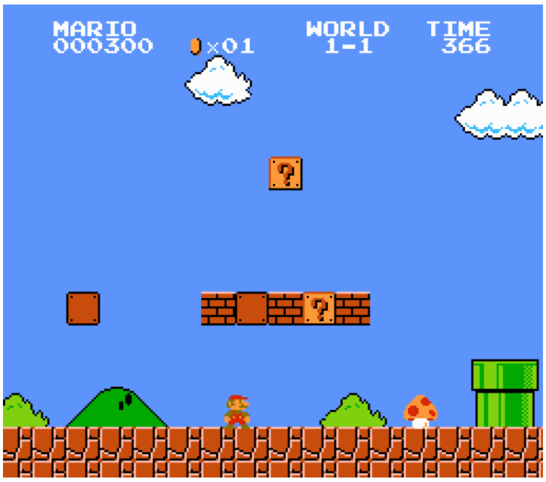


Figura 2.1: *Captura del videojoc Super Mario Bros. (NES, 1985)*

Des d'aquesta segona oportunitat hi ha hagut molt escepticisme sobre si el píxel art només serà una moda passatgera o ha arribat per quedar-se definitivament. Aquesta qüestió va ser plantejada l'any 2012 per l'artista Oliver Huard, qui va defensar el valor històric i sentimental que implica aquesta forma d'art:

> *"La part sentimental exerceix un paper determinant en aquest retorn del pixel art. Pels usuaris que han arribat als trenta anys la nostàlgia de les sessions de joc amb les seves velles consoles i jocs populars de la època. Això es part d'un moviment de "renaixement" que s'observa en moltes àrees: sèries de televisió, joguines, mobles, etc."*

Figura 2.2: *Captura del videojoc Hyper Light Drifter (PC, 2016)*

Com bé diu Huard a la cita anterior, aquest retorn té una gran càrrega emocional, el que ha fet que el *pixel art* hagi adquirit una identitat i uns estàndards que als seus inicis no tenia. Aquestes normes són el que han fet que aquest estil tingui una personalitat pròpia que el diferencia de la resta.

2.1.1 Tècniques de Pixel Art

2.1.1.1 Mida i Escales

Una de les grans diferències del *pixel art* respecte a la resta d'estils artístics és que es treballa en mides molt petites, ja que, com el seu propi nom indica, es basa en l'ús òptim dels píxels. Aquesta limitació, amb un origen purament tècnic, obliga a que cada unitat de píxel tingui les mateixes proporcions.

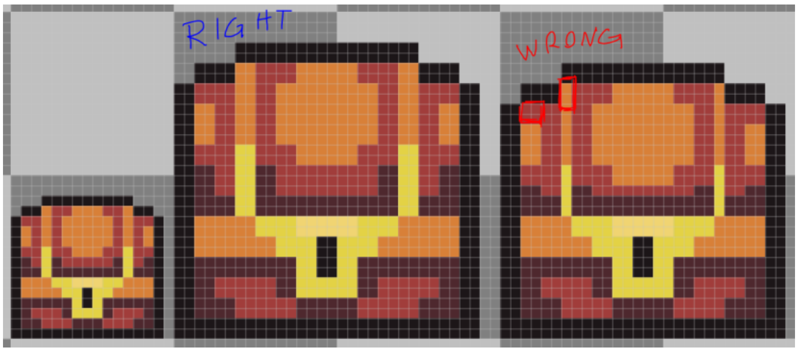


Figura 2.3: *Comparativa de pixel art reescalat de manera correcta i incorrecta al programa Aseprite*

A la figura 2.3 es pot observar el problema que pot sorgir quan una imatge *pixel art* es reescala malament. Per evitar l'aparició de píxels deformats és necessari escalar les imatges amb valors sencers, així el píxel multiplicarà la seva mida en les unitats que l'usuari desitgi.

Com a exemple, a la figura anterior la imatge del centre ha reescalat la imatge per 2, per tant els píxels s'han duplicat. La imatge de la dreta ha sigut escalada per 1.07, derivant en una imatge amb píxels imperfectes.

2.1.1.2 Pixel Perfect

Degut a la forma quadrada del pixel és normal que apareguin cantonades molt recargades a l'hora de fer corbes. Per aconseguir el que es coneix com una línia *pixel perfect* s'ha de procurar que aquesta només contingui una unitat de píxel, el qual s'aconsegueix eliminant els quadrats que generen soroll al dibuix.

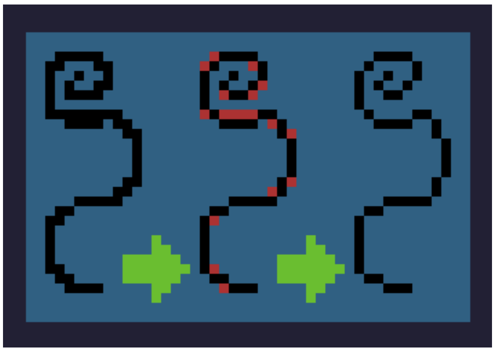


Figura 2.4: *Exemple dels passos a seguir per aconseguir una línia pixel perfect al programa Aseprite*

El concepte *pixel perfect* es podria descriure com a la representació més fidel possible al concepte de "píxel". És per això que no es limita als dibuixos fets manualment.

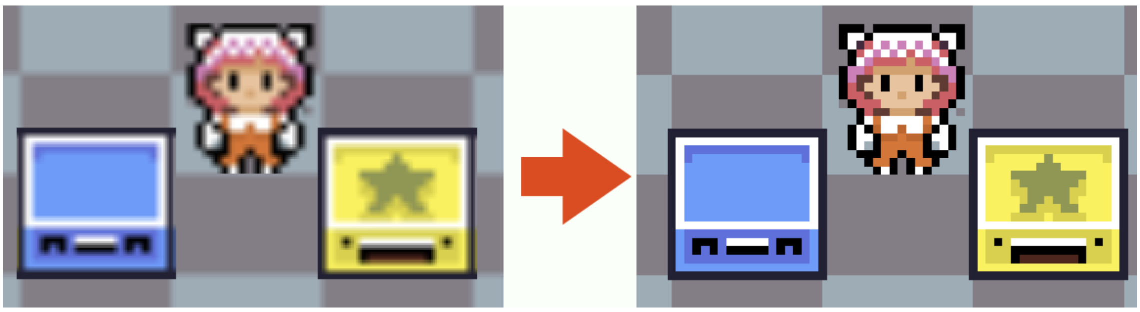


Figura 2.5: *Captures del motor Unity on es passa d'una imatge borrosa a una imatge pixel perfect*

A la figura 2.5 s'aprecia, en primera instancia, una imatge borrosa degut als filtres que aplica per defecte, en aquets cas, el motor Unity. Per aconseguir que la imatge mantingui la forma real del pixel i així aconseguir una estètica *pixel perfect* s'han d'ajustar un seguit de valors per poder obtenir un resultat com el de la imatge de la dreta.

2.1.1.3 Colors i Shading

De la mateixa manera que en el cas de la mida, els colors juguen un paper clau en aquest estil de representació gràfica. Les limitacions tècniques del passat obligaven a utilitzar paletes de color molt optimitzades, costum que amb el pas del anys s'ha mantingut, fent que els *pixel artists* normalment limitin els colors i la quantitat de *shades* o ombres de cada tonalitat.

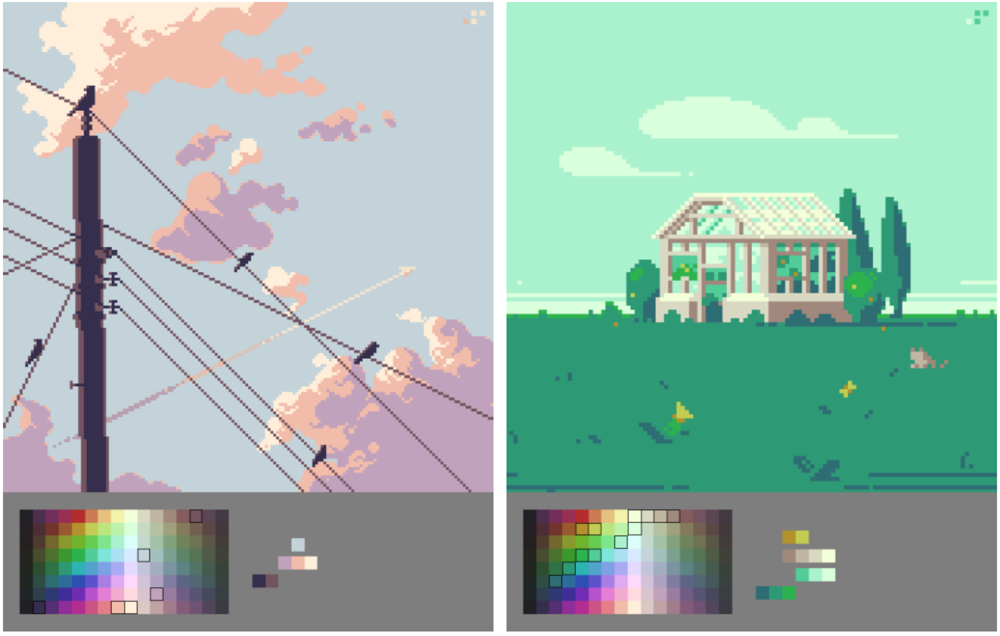


Figura 2.6: *Exemples d'il·lustracions pixel art amb paletes de color limitades*

A la figura 2.6 s'observa com, tot i comptar amb paletes de 6 i 12 colors respectivament, s'aconsegueix un resultat de gran qualitat. Això és degut al correcte ús de les ombres i el contrast entre els colors escollits.

Figura 2.7: *Ànec Pixel Art amb diferents tonalitats aconseguides amb la tècnica de Hue Shifting*

Tècniques d'ombrejat o *shading* són les que marquen la diferència a l'hora de realitzar un bon *pixel art*. La base d'un bon *shading* és un bon coneixement teòric del funcionament de la llum, però també hi ha pràctiques més específiques com el *hue shifting* que donen un grau més de profunditat realitzant una quantitat molt reduïda de canvis. En aquest cas, aquesta tècnica consisteix en variar els colors de la paleta, buscant un contrast alt entre les diferents tonalitats afegint colors complementaris a les ombres.

2.1.1.4 Antialiasing i Dithering

Per fer una transició de colors més fluida hi ha diferents tècniques que apliquen els *pixel artists*, de les quals dues de les més conegudes són l'*antialiasing* i el *dithering*:

L'*antialiasing* consisteix en l'addició de subtonalitats a les vores de zones amb colors molt diferents. Mentre que és una de les pràctiques amb resultats més nets normalment suposa un increment considerable de la paleta si la peça compta amb molts colors diferents.

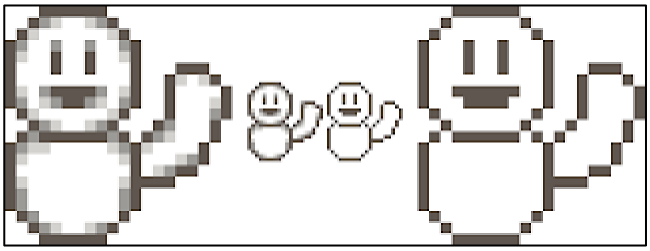


Figura 2.8: *Comparativa entre dos dibuixos, amb i sense l'aplicació de la tècnica d'antialiasing respectivament*

El *dithering*, per l'altra banda, és una tècnica totalment contrària a l'anterior. En aquest cas, a partir de diferents patrons a gust de l'artista, es passa d'una tonalitat a una altra totalment diferent sense fer servir cap color intermedi.

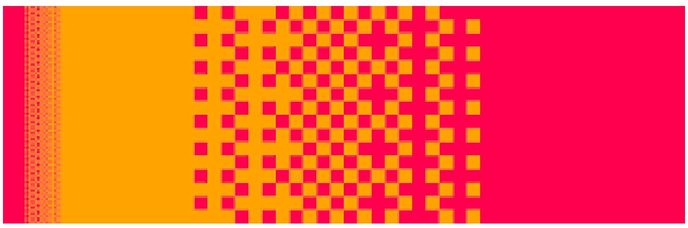


Figura 2.9: *Degradat de color mitjançant la tècnica de dithering*

2.2 Estudi de Mercat

Una vegada analitzades les bases del *pixel art* farà falta un estudi de les solucions que hi ha ara mateix al mercat. Tot i que l'objectiu del projecte és desenvolupar una eina d'efectes especials, també pot ser interessant observar les característiques d'alguns dels editors d'imatge disponibles.

També s'estudiaran les opcions que donarà Unity en relació al *pixel art* i s'acabarà amb una anàlisi de l'eina que competiria directament amb la d'aquest projecte, Pixel FX Designer.

2.2.1 Software d'edició manual de Pixel Art

El software d'edició de *pixel art* compta amb eines molt variades. Entre aquestes eines, obviant les que es troben a editors d'imatge convencionals, es poden destacar:

- **Llapis amb filtre *pixel perfect***: Permet a l'usuari dibuixar línies aplicant automàticament l'efecte *pixel perfect*.

- **Opcions simples d'animació:** Amb l'ajut d'un *timeline* dona l'opció de crear diferents fotogrames de fàcil accés per poder crear animacions de manera ràpida.

- **Configuracions i paletes de color preestablertes:** Les configuracions predefinides que ofereixen aquest tipus de programes estan enfocades a limitacions de mides i paletes utilitzades exclusivament en el *pixel art*.

Les característiques que ofereixen la gran majoria són molt semblants entre sí.[4] Per aquesta raó, s'han destacat les dues eines d'edició més populars d'aquest sector: Aseprite i PyxelEdit.

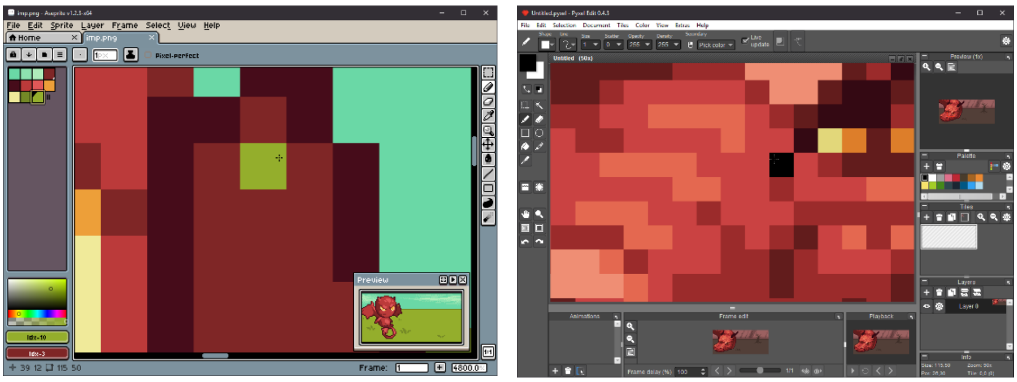


Figura 2.10: *Aseprite i PyxelEdit en funcionament*

[4] Conclusió extreta a partir de l'anàlisi de l'oferta de software observat en Lospec <https://lospec.com/pixel-art-software-list> (Consultat el 13/03/2019)

Aseprite[5]	**PyxelEdit**[6]
Preu: 14,99€	Preu: 9,00$ (7,99€ aprox.)
UI feta en *Pixel Art*	UI simple i intuïtiva.
Eines d'animació com *onion skin*, velocitat de cada *frame* i configuració del *loop.*	La millor selecció d'eines de tilesets del mercat.
Gran quantitat d'opcions d'exportació de *spritesheets.*	Versió gratuïta disponible.

Taula 2.1: *Comparativa de característiques entre Aseprite i PyxelEdit*

2.2.2 Unity

Unity és un dels motors de jocs més populars actualment. Va ser llançat al mercat l'any 2005, i des de llavors ha anat rebent actualitzacions fins arribar a la versió actual, la 2018.3.8.

Mentre que el seu competidor directe, Unreal Engine, es centra en el desenvolupament de jocs 3D, Unity compta amb una gran quantitat de propietats preparades per la producció de jocs 2D, raó per la qual s'ha escollit aquest motor a l'hora de plantejar el projecte.

D'entre aquestes característiques, aquestes són algunes de les que ajudaran en l'objectiu de generar l'*asset* d'efectes especials *pixel art*.

2.2.2.1 Particle System

Unity ja compta amb un sistema de generació de FX integrat conegut com a *Particle System*, traduït a "sistema de partícules". Aquest sistema inclou un seguit d'opcions que permeten a l'usuari crear efectes especials de molts tipus diferents, de les quals es poden destacar:

Emissor	Variable que determina el rati de partícules generades i la distància que poden recórrer aquestes.
Shape (Forma)	Valor que donarà la forma gràfica de la partícula.
LifeTime (Temps de Vida)	Moltes de les variables es poden limitar i modificar al llarg del "temps de vida" de la partícula, com per exemple la forma, color o velocitat.
Collision (Col·lisió)	Opció que permet controlar la col·lisió i les físiques de les partícules amb l'escenari.
Lights (Llums)	Mòdul que permet incloure llums en temps real a les partícules.
Trail (Rastre)	Valors que afegeixen un rastre visual a les partícules.

Taula 2.2: *Característiques del Particle System de Unity*[7]

[5] Dades extretes del lloc web oficial d'Aseprite < https://www.aseprite.org/> (Consultat el 13/03/2019)
[6] Dades extretes del lloc web oficial de PyxelEdit <https://pyxeledit.com/> (Consultat el 13/03/2019)
[7] Dades extretes de la documentació oficial de Unity
< https://docs.unity3d.com/Manual/ParticleSystems.html> (Consultat el 13/03/2019)

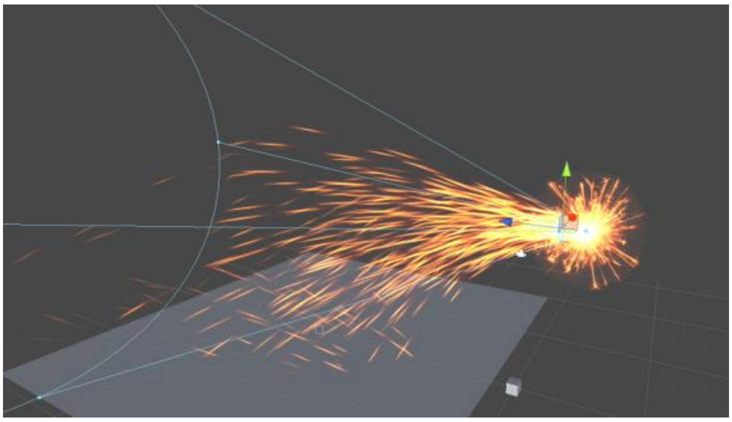


Figura 2.11: *Espurnes creades mitjançant el Particle System de Unity*

2.2.2.2 Shaders

Els *shaders* són programes que determinen la manera com es pinta un element en pantalla.

Ja que Unity compta amb un sistema sòlid de generació de partícules, la creació d'un *shader* que transformi a estil *pixel art* les imatges mostrades pel motor seria de gran ajuda en aquest projecte. Això és possible gràcies a les opcions de personalització que ofereix el sistema de *shaders* d'aquest motor, arribant a tenir resultats com el de la següent figura:

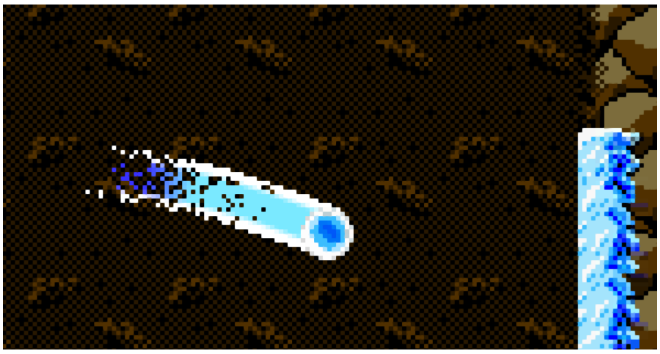


Figura 2.12: *Partícula amb shaders estil pixel art*

2.2.2.3 Unity Asset Store

L'objectiu final del projecte és llançar el producte desenvolupat a l'Asset Store de Unity. Aquesta botiga online permet als usuaris pujar *assets* de qualsevol tipus, des de models 3D preparats per utilitzar a un joc fins a editors visuals d'intel·ligència artificial.

Pel que respecta al camp de treball d'aquest projecte, l'oferta és molt limitada[8], ja que només hi ha uns pocs *assets* que ofereixen partícules *pixel art*. En addició a això, aquests *assets* el que inclouen són partícules prefabricades amb opció a personalització, així que la eina resultant d'aquest projecte seria la primera en cobrir aquesta necessitat a Unity.

[8] Conclusió extreta a partir de l'anàlisi de la següent cerca dins de l'Asset Store de Unity <https://assetstore.unity.com/search?k=pixel+art+fx&order_by=relevance&q=pixel&q=art&q=fx&rows=42> (Consultat el 13/03/2019)

2.2.3 Pixel FX Designer

Pixel FX Designer va ser llançat a l'estiu de 2018, desenvolupat per Davit Masià i Manuel Bolaños, sent l'únic software al mercat que realitza la mateixa funció que aquest projecte. Funciona de manera independent a cap motor i conté les següents característiques:

Preu de 19,95$ (16,79€)	Paletes fins a 256 colors
Disponible a Steam i itch.io	*Timeline* d'animació
Dithering personalitzat	Dibuix manual del camí de la partícula
Loop d'animació polit i continu	Sistema de capes

Taula 2.3: *Característiques de Pixel FX Designer*[9]

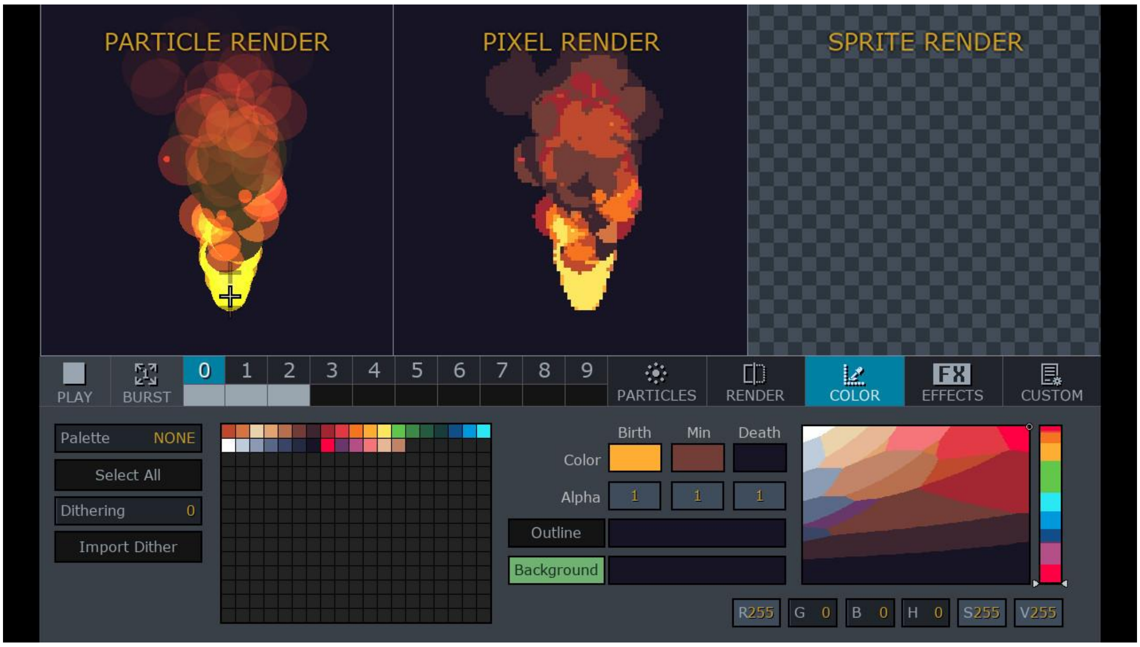


Figura 2.13: *Captura del software Pixel FX Designer*

[9]Dades extretes del lloc web oficial de Pixel FX Designer
<https://codemanu.itch.io/particle-fx-designer> (Consultat el 13/03/2019)

3. Gestió del projecte

3.1 Procediment i Eines per al seguiment del projecte

3.1.1 Fases de producció i diagrama GANTT

El desenvolupament d'aquest *asset* es dividirà en les següents fases de producció:

Preproducció

Durant aquesta fase es planificarà i dissenyarà tot el necessari de cara al desenvolupament de l'*asset*.

- **Disseny de la funcionalitat -** Setmana 1 (18 de març fins 25 de març):
  Per començar és necessari tenir clares les característiques de l'*asset* que es vol desenvolupar. Durant aquesta tasca es crearà una documentació on s'explicaran totes les funcionalitats que es volen implementar.

- **Disseny de la UI** – Setmana 2 (25 de març fins 1 d'abril):
  Una vegada decidida la funcionalitat es passarà a dissenyar la interfície de l'*asset*, centrant-se sobretot en aconseguir una UI *user-friendly*.

- **Estudi de les llibreries de Unity** – Setmanes 3 i 4 (1 d'abril fins 15 d'abril):
  Per poder realitzar la producció de manera satisfactòria es realitzarà un estudi de totes les eines i llibreries de Unity que poden facilitar el desenvolupament de l'*asset*.

Producció

Aquesta fase es la que més càrrega de feina comporta, englobant tot el desenvolupament de l'*asset* fins la seva versió final.

- **Implementació de la UI** – Setmana 5 (15 d'abril fins 22 d'abril):
  Per començar es desenvoluparà la interfície base de treball, a la qual s'aniran afegint funcionalitats a mesura que cada part de l'*asset* es vagi acabant.

- **Shader Base** – Setmanes 6 i 7 (22 d'abril fins 6 de maig):
  La primera funcionalitat a desenvolupar és el sistema de filtre a *pixel art,* el qual farà servir tant eines del motor com shaders personalitzats. Es dedicaran dues setmanes ja que és un dels punts clau de l'*asset*. En entregar la rúbrica 2 s'haurà desenvolupat fins aquest punt.

- **Opcions d'exportació** – Setmana 8 (6 de maig fins a 13 de maig):
  Per al MVP s'afegiran funcionalitats d'exportació, tenint així un producte que ja pugui funcionar tant dins del motor com en produccions externes.

- **Release MVP** – Setmana 9 (13 de maig fins a 20 de maig):
  Una vegada acabades les funcionalitats més bàsiques de l'*asset* es farà una *release* del MVP (Producte Mínim Viable).

- **Paletes** – Setmana 10 i 11 (20 de maig fins a 3 de juny):
  La primera funcionalitat que s'afegirà de cara al producte final serà la personalització de les paletes dels FX generats. Es dedicarà una setmana a crear la interfície i funcionalitats necessàries.

- **Antialiasing** – Setmana 12 (3 de juny a 10 de juny):
  Aquesta setmana es dedicarà a la implementació de les opcions d'antialiasing.

- **Dithering** – Setmana 12 (3 de juny a 10 de juny):
  De la mateixa manera, es dedicarà una setmana a les opcions de *dithering* dissenyades prèviament. S'ha decidit realitzar aquesta tasca conjuntament amb la d'*antialiasing* ja que es preveu que la càrrega de treball serà més reduïda que la resta.

- **Release Beta** – Setmana 13 (10 de juny fins 17 de juny):
  Durant aquesta setmana es llançarà la versió beta de l'*asset*, que inclourà totes les funcionalitats planejades.

- **Testing** – Setmana 9 (13 de maig fins a 20 de maig) i Setmana 13 (10 de juny fins 17 de juny):
  Es realitzaran sessions de testeig per recollir feedback durant les *releases* de la versió MVP i Beta.

- **Polish** – Setmana 9 (13 de maig fins a 20 de maig) i Setmana 13 (10 de juny fins 17 de juny):
  Paral·lelament, durant les mateixes dates es polirà l'*asset* a partir dels problemes sorgits i el feedback recollit.

Release

- **Llançament a l'Asset Store** – Setmana 14 (17 a 24 de juny):
  Es planeja llançar l'*asset* durant aquesta setmana per així poder tenir tot llest de cara al depòsit del TFG el dia 25 de juny.

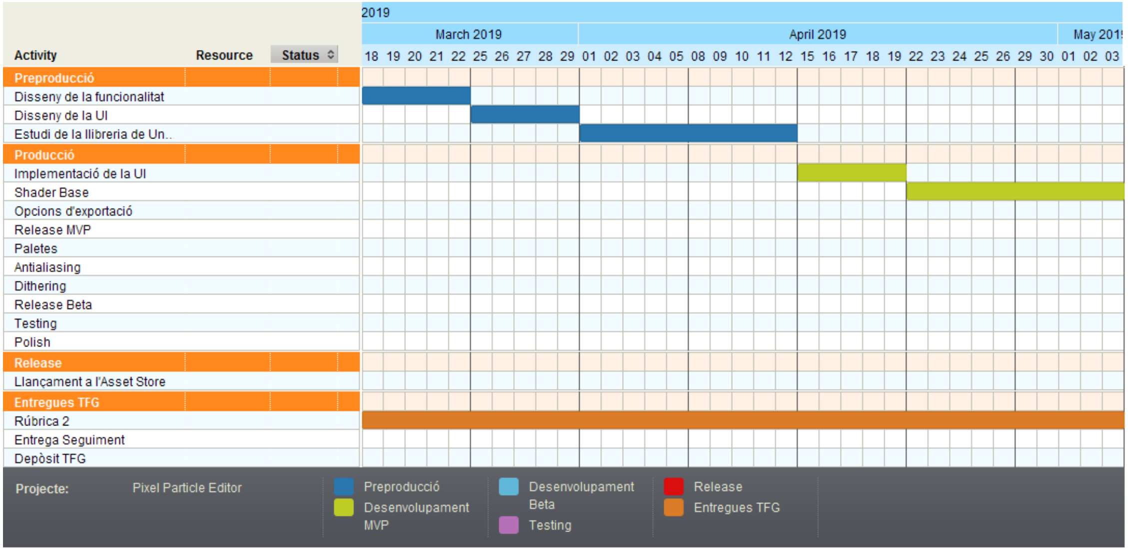


Figura 3.1: *Versió 2 del Diagrama Gantt durant la entrega de la Rúbrica 2*

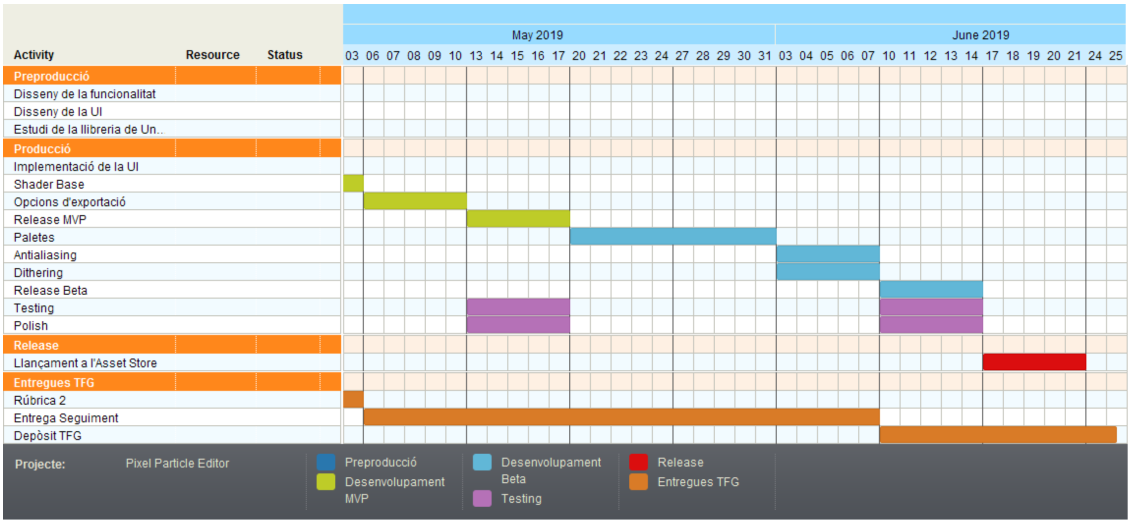


Figura 3.2: *Versió 2 del Diagrama Gantt durant la entrega de seguiment i el dipòsit del TFG*

3.1.2 Eines de seguiment: HacknPlan i GitHub

Per realitzar el seguiment d'aquest projecte es faran servir les eines HacknPlan i Github.

HacknPlan és un lloc web que permet una fàcil gestió de projectes i de les seves tasques. En l'apartat de metodologia s'explicarà la forma de treball que es seguirà, i amb HacknPlan serà molt fàcil organitzar les tasques que sorgiran.

GitHub, d'altra banda, serà l'eina a la qual es pujarà el codi del projecte. És un servei basat en Git que permet crear un repositori i tenir un control dels canvis que es van realitzant al codi mitjançant *commits*, els quals serien un punt de control on es puja certa secció del codi i s'emmagatzemen tots els canvis realitzats.

3.2 Eines de validació

Com s'ha especificat al punt anterior, per validar el funcionament de l'*asset* es realitzaran dues sessions de *testing*.

El *testing* es basarà en entregar la versió que s'hagi de validar a un seguit d'usuaris i durant un temps determinat provaran el producte per així poder donar feedback en acabar la sessió.

L'objectiu d'aquestes sessions serà, d'una banda, observar el funcionament correcte de l'*asset* i detectar la possible aparició de *bugs.* Per l'altra banda s'estudiarà el feedback recollit per millorar la experiència d'usuari i plantejar possibles funcionalitats que no s'hagin tingut en compte.

Els usuaris que participin al *testing* rebran una enquesta realitzada amb Google Forms, a partir de la qual valoraran la experiència i proposaran canvis i/o millores per l'*asset*. Les enquestes que realitzaran els usuaris s'adjuntaran als annexos del treball en cas de que es vulguin consultar, de la mateixa manera els resultats de les sessions de validació seran valorats al punt 5.3 d'aquest document.

Google Forms, l'eina escollida per realitzar les enquestes, és una de les plataformes que ofereix Google per realitzar consultes de manera senzilla, amb opcions de quantificar i comparar els resultats obtinguts en format de gràfiques i taules de càlcul.

3.3. DAFO

Amb l'objectiu de tenir una visió general del transcurs del projecte s'ha fet una anàlisi dels punts positius i negatius d'aquest, tant des d'un punt de vista intern com extern:

	Positius	Negatius
Origen Intern	**Fortaleses** - Oferta única a l'Asset Store de Unity. - Coneixement previ en *Pixel Art*, el qual ajuda en definir la qualitat desitjada.	**Debilitats** - Falta de coneixement en programació de *shaders*, el qual pot alentir la producció. - Temps de desenvolupament limitat.
Origen Extern	**Oportunitats** - Opció a realitzar noves eines *pixel art* per a Unity a partir del *shader* creat. - Possibilitat de desenvolupar una versió externa de l'*asset* que pugui competir amb Pixel FX Designer.	**Amenaces** - Ja existeix un software que realitza la mateixa funció fora de Unity. - Els artistes poden preferir realitzar els seus efectes especials de manera totalment manual.

Taula 3.1: Anàlisi DAFO

3.4. Riscos i pla de contingències

Tot projecte està subjecte a possibles problemes durant el seu desenvolupament, raó per la qual s'ha fet un estudi dels riscos als que es pot exposar aquest treball, juntament amb les solucions que es proposen:

Risc	Solució
- Falta de coneixement en programació de *shaders*	- Estudiar i analitzar la documentació de *shaders* de Unity durant la fase de preproducció, per així arribar preparat al desenvolupament. - Aplicar els coneixements de *pixel art* per obtenir bons resultats amb l'aplicació dels estudis fets prèviament.
- Falta de temps per desenvolupar el projecte	- Prioritzar el desenvolupament de les funcions clau de l'*asset* per poder tenir un producte funcional al final del projecte. - Subdividir les tasques programades per tenir una visió precisa del temps a dedicar, i modificar la càrrega de treball entre tasques si és necessari.
- Els FX resultants poden no tenir la qualitat esperada	- Realitzar sessions de testing amb artistes especialitzats en *pixel art* per poder recollir un feedback que ajudi a la millora de l'*asset*. - Estudiar durant la fase de preproducció el funcionament de *shaders* i eines que pintin gràfics en *pixel art.*

Taula 3.2: *Anàlisi de riscos i pla de contingències*

3.5. Anàlisi inicial de costos

Per calcular les despeses que suposaria un projecte com aquest s'han tingut en compte els següents factors:

- Està desenvolupat per una sola persona, així que s'ha estimat el sou d'un artista junior a mitja jornada[10].

- Per realitzar l'*asset* farà falta estudiar l'oferta que hi ha al mercat, pel qual s'ha afegit la compra de software com Pixel FX Designer i Aseprite.

- El temps de treball total són 4 mesos, per tant el càlcul es farà en relació a aquest temps.

Després de realitzar els càlculs, s'estima que el cost aproximat d'aquest projecte seria d'uns 2.825,78€.

	Març	Abril	Maig	Juny	Total
Costos Directes	660,28 €	628,50 €	628,50 €	628,50 €	**2.545,78 €**
Personal	588,50 €	588,50 €	588,50 €	588,50 €	**2.354,00 €**
Programador Júnior	588,50 €	588,50 €	588,50 €	588,50 €	**2.354,00 €**
Software	31,78 €	- €	- €	- €	**31,78 €**
Pixel FX Designer	16,79 €	- €	- €	- €	**16,79 €**
Aseprite	14,99 €	- €	- €	- €	**14,99 €**
Unity	- €	- €	- €	- €	**- €**
Visual Studio	- €	- €	- €	- €	**- €**
Equip de treball	40,00 €	40,00 €	40,00 €	40,00 €	**160,00 €**
Ordinador (Depreciació)	40,00 €	40,00 €	40,00 €	40,00 €	**160,00 €**
Costos Indirectes	70,00 €	70,00 €	70,00 €	70,00 €	**280,00 €**
Electricitat	30,00 €	30,00 €	30,00 €	30,00 €	**120,00 €**
Internet	40,00 €	40,00 €	40,00 €	40,00 €	**160,00 €**
Cost Total	730,28 €	698,50 €	698,50 €	698,50 €	**2.825,78 €**

Taula 3.3: *Estimació de costos del projecte*

[10] Estimació feta a partir de les dades extretes d'indeed.es <https://www.indeed.es/salaries/Programador/a-junior-Salaries?period=monthly> (Consultat el 14/3/2019)

4. Metodologia

La metodologia escollida pel desenvolupament d'aquest projecte és el *scrum*. El *scrum* és una metodologia àgil que es caracteritza per treballar en base a petits objectius, coneguts com a *sprint*.

Com s'ha pogut observar a l'apartat 3.1.1, el projecte ja està plantejat d'aquesta manera, treballant amb *sprints* d'una setmana i de dues setmanes en els casos de més càrrega de treball.

Aquests es seguiran a partir de HacknPlan, tal i com s'ha dit a l'apartat 3.1.2, subdividint els objectius del *sprint* en petites tasques. Dins de l'eina, les tasques es poden trobar en quatre estats diferents:

- **PLANNED:** La tasca està pendent de ser realitzada.
- **IN PROGRESS**: La tasca està en progrés.
- **TESTING:** La tasca està pendent de validació.
- **COMPLETED:** La tasca ha sigut completada.

Cada tasca comptarà amb els següents valors:

- **Category:** Apartat de *l'asset* on va destinada la tasca. Per exemple, les tasques destinades a la programació de *shaders* tindran la categoria "Shader".
- **Estimated Time:** Temps de treball estimat per acabar la tasca.
- **Logged Time:** Temps de treball dedicat a la tasca. Al final de cada *sprint* es farà una valoració dels temps dedicats i estimats per analitzar futures desviacions en les càrregues de treball estimades a l'apartat 3.1.2.

Ja que és un treball individual no hi haurà *stand-ups* diàries, una pràctica que es duu a terme en el *scrum* on tots els membres de l'equip repassen diàriament la feina feta. En aquest cas el control del treball el durà l'alumne a diari, mentre que el tutor revisarà l'estat de les tasques dins de la seva disponibilitat.

Les fases de treball d'aquest projecte també estan dividides, en aquest cas segons el tipus de feina en realitzar en cada etapa:

- **Preproducció:** Es realitzarà tota la feina de disseny i conceptualització de l'*asset*
- **Producció:** Etapa d'implementació, on la feina serà purament tècnica.
- **Release:** Fase final amb el producte desenvolupat. Es centrarà en el llançament de l'*asset* al mercat.

Les tasques que, per causes de temps o de dificultat, no es puguin realitzar aniran a l'apartat de *Backlog,* on quedaran guardades per la seva futura realització en *sprints* posteriors o una vegada el projecte hagi finalitzat i l'*asset* estigui penjat a l'Asset Store de Unity.

A l'annex 8.2 s'inclourà una quantificació del temps dedicat a cada tasca realitzada i el temps estimat d'aquestes.

5. Desenvolupament

5.1 Preproducció

Per començar amb el desenvolupament d'aquest *asset* fa falta dissenyar tot el que engloba el producte a crear. Aquest disseny s'ha dividit en les dues parts clau que engloba qualsevol disseny de software: el *back-end* i el *front-end*.

El *back-end* és el nucli del programa, és a dir, tot el codi que realitza les funcions per les quals el programa ha estat creat. Tenint en compte que aquest *asset* es desenvoluparà amb el motor Unity, a l'hora de realitzar aquest apartat seran clau les llibreries de *Particle Systems*, *Shaders* i *Camera* del motor esmentat.

El *front-end*, per l'altra banda, és la part superficial del desenvolupament, on es programaran els elements que es mostraran en pantalla amb els quals l'usuari interaccionarà. Aquí les llibreries d'Editor facilitaran el desenvolupament de la *UI* de l'*asset*.

5.1.1 Back-End (Disseny de les funcionalitats)

Les funcionalitats d'aquest *asset* estaran basades en les tècniques de *pixel art* analitzades a l'apartat de "Estat de l'Art", les quals han estat dividides en els següents apartats:

5.1.1.1 Base

La funcionalitat base de l'*asset* és aconseguir que les partícules creades per l'usuari es renderitzin en estil *pixel art*. Per aconseguir aquest resultat es faran servir, d'una banda, càmeres de Unity configurades en baixa resolució i, per l'altra, *shaders* per acabar de polir les imatges a les càmeres configurades.

5.1.1.2 Creació de Partícules

Mentre que Unity ja compta amb una interfície de creació de partícules molt competent, en aquest projecte s'afegirà una opció simplificada per a usuaris més novells o que només vulguin les opcions més adequades per l'estil *pixel art*. Pels usuaris que prefereixin treballar amb l'editor de partícules per defecte també s'afegirà una opció d'importar partícules dins l'editor.

Per treballar amb comoditat també s'implementarà un sistema de capes molt semblant als de *Pixel FX Designer* i *Aseprite.*

Totes aquestes funcionalitats s'aconseguiran amb l'ús i modificació dels valors de cada sistema de partícules, els quals seran modificats directament des del codi de la *UI*.

Els valors que podrà modificar l'usuari són:

- **<u>Basic:</u>**

- **Texture List:** textura que donarà forma a la partícula. L'usuari podrà afegir textures personalitzades.
- **Duration (Float):** duració de cada cicle de la partícula.
- **Looping (Bool):** determina si la partícula compta amb un sol cicle o amb cicles infinits.

- **Start Delay (Float):** determina el temps d'espera abans de començar el cicle de la partícula.
- **Lifetime (Float):** valor que determina els segons que cada partícula està en funcionament.
- **Particle Speed (Float):** velocitat de cada partícula.
- **Animation Speed (Float):** velocitat a la que es reproduirà l'animació de la partícula.
- **Size (Float):** mida de la partícula.
- **Size Increment (Llista):** valor que determinarà la variació de la mida de la partícula durant cada cicle.
- **Gravity (Float):** valor que determina com afecta la gravetat a cada partícula.
- **Max Particles (Int):** nombre màxim de partícules en pantalla.
- **Particles per second (Int):** nombre màxim de partícules emeses cada segon.

- **Shape:**

- **Active (Bool):** determina si la partícula es veu afectada per una forma específica.
- **Type (Llista):** determina quina forma afectarà la emissió de partícules en cas que el paràmetre anterior estigui actiu.
- **Cone Angle (Float):** en cas que la forma sigui un con, l'usuari pot definir l'angle d'obertura d'aquest.
- **Position (XYZ Value):** valor que determina la posició d'origen de la partícula
- **Rotation (XYZ Value):** valor que determina la rotació del punt d'origen de la partícula
- **Scale (XYZ Value):** valor que determina la mida del punt d'origen de la partícula

5.1.1.3 Colors

Com s'explica a apartats anteriors del treball, els colors juguen una part clau en el *pixel art*. L'usuari tindrà a la seva disposició les paletes de colors utilitzades a cada capa de partícules i podrà modificar-les amb paletes predefinides o personalitzades.

Per aquesta funcionalitat es modificaran tant els valors de color del sistema de partícules com els *shaders* destinats a la modificació de colors.

Els atributs que seran modificats per l'usuari són:

- **Color RGB:** determina el color del punt de la paleta escollit per l'usuari.
- **Palettes (Llista de textures):** sent cada textura una paleta, l'usuari podrà obrir, carregar i crear noves paletes.
- **Blending Mode (Llista de int):** llista amb els tipus de pintat a pantalla disponibles per l'usuari.

5.1.1.4 Efectes Especials

En aquest apartat l'usuari tindrà al seu abast diferents opcions per personalitzar els efectes de *dithering* i *antialiasing* de les seves partícules. Ambdós efectes es desenvoluparan mitjançant *shaders*.

Els dos efectes comptaran amb el mateix tipus d'atributs personalitzables:

- **Active (Bool):** determina si està actiu l'efecte en qüestió.
- **Instensity (Float):** determina la intensitat de l'efecte aplicat, sent 0.0f el mínim i 1.0f el màxim.

A més, també s'afegirà la opció d'incloure un *outline*, és a dir, una línia de contorn del color que desitgi l'usuari. Els paràmetres que determinaran aquest efecte seran:

- **Active (Bool):** determina si està actiu l'*outline.*
- **Color RGB:** color de l'*outline.*

5.1.1.5 Càmera

Ja que la finestra de treball de la interfície serà limitada es donarà la opció de modificar la posició i el camp de visió de les càmeres que renderitzaran les partícules. A més, s'inclouran punts de vista predefinits per facilitar la creació de partícules amb vistes laterals, *top down*[11] i isomètriques. Quant a codi, aquesta part es realitzarà únicament amb l'ajut de la llibreria de càmeres de Unity.

L'input disponible per l'usuari serà el següent:

- **View List (Llista de Int):** llista amb els tipus de vistes predefinides disponibles.
- **Apply View (Trigger):** botó per activar la vista seleccionada.
- **Camera Position (Valor XY):** posició de la càmera. Es prescindeix del valor Z ja que es treballarà en 2D.
- **Camera Angle (Valor XYZ):** angle de visió de la càmera.
- **Field of View (Float):** camp de visió de la càmera.

[11] Les vistes *top down* són aquelles en les que l'angle de la càmera està lleugerament inclinada en l'eix Y però no hi ha possibilitat d'apreciar l'eix Z, com és en el cas de la vista isomètrica. The Legend of Zelda: A Link to the Past o Pokémon són exemples de videojocs que han fet servir aquest punt de vista.

5.1.1.6 Importació

Un dels objectius del treball és oferir un producte que puguin fer servir tant usuaris novells com usuaris més avançats. Per aquesta raó s'ha considerat que hi haurà una part d'usuaris que preferiran crear les seves partícules directament des de l'editor per defecte de Unity, el qual compta amb un catàleg molt més ampli de funcionalitats. Degut a aquest fet, s'inclouran opcions d'importació de partícules existents, ja siguin creades prèviament amb aquest *asset* o amb l'editor original.

Les funcionalitats disponibles per l'usuari seran:

- **Import List (Llista de Game Objects):** llista de partícules a importar.
- **Overwrite (Bool):** valor que determinarà si la partícula importada substituirà l'actual.
- **Layer to Import (Llista de capes):** capa a la qual s'importarà en cas que el valor anterior no estigui actiu.
- **Import (Trigger):** botó per importar la partícula amb els ajustos seleccionats.

5.1.1.7 Exportació

Per acabar, l'usuari podrà accedir a una de les funcionalitats clau d'aquest *asset*, l'exportació. Ja que cada desenvolupador pot tenir un objectiu diferent, amb l'ús d'aquest software s'han considerat diferents alternatives d'exportació. Aquestes alternatives són les següents:

- **Game Object Export:** exportació directe al motor Unity.
- **GIF Export:** exportació en forma de GIF animat.
- **Sprite Sheet Export:** exportació en forma de textura.

L'input d'aquesta funcionalitat serà:

- **Frames to Export (Int):** quantitat de fotogrames que tindrà l'exportació.
- **Animation Time (Float):** temps en segons de l'animació exportada.
- **Export (Trigger):** botó per exportar la partícula creada amb els ajustos seleccionats. Hi haurà un botó per cada tipus d'exportació.

5.1.2 Llista de funcionalitats i prioritats

Una vegada plantejades totes les funcionalitats desitjades, s'ha realitzat una taula a la qual es mostren de manera més clara, afegint una prioritat a cadascuna d'aquestes de cara al futur desenvolupament de l'*asset*.

Les prioritats plantejades són les següents:

- **High**: Prioritat alta, funcionalitat essencial de l'*asset*
- **Normal**: Prioritat mitja, funcionalitat desitjada per l'*asset*, però no essencial
- **Low**: Prioritat baixa, funcionalitat plantejada per complementar les anteriors

Feature	Priority
Particle Editing	High
Layer System	High
Palette Editing	High
Color Blending	Normal
Dithering	Normal
Antialiasing	Normal
Outline	Normal
Camera Settings	Low
Import Options	High
Game Object Export	High
Sprite Sheet Export	Normal
GIF Export	Normal

Taula 5.1: *Llista de funcionalitats i prioritats*

5.1.3 Front-End (Disseny de la UI)

5.1.3.1 Anàlisi de la UI de la competència

Per crear la interfície d'aquest *asset* s'han analitzat prèviament els programes de la competència, els quals segons el seu impacte són Aseprite i Pixel FX Designer. Aseprite ha sigut escollit per ser el programa *pixel art* amb més us del mercat, mentre que Pixel FX Designer s'ha tingut en compte per ser l'unic programa que resol en gran mesura el problema plantejat per aquest TFG.

En les figures següents es poden observar captures de pantalla d'ambdós programes, dels quals s'han destacat elements que inclouen els esmentats *softwares.*

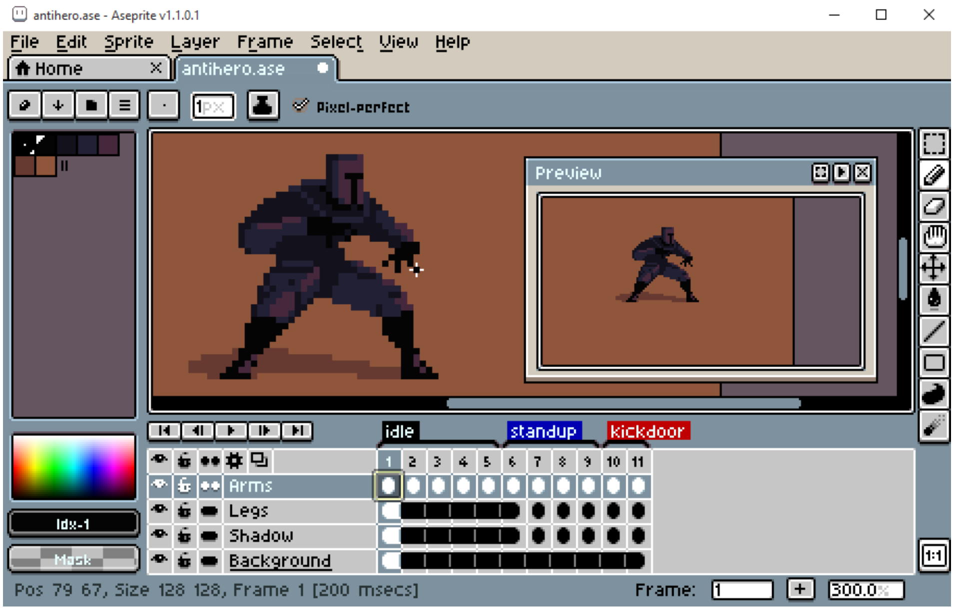


Figura 5.1: *Captura de pantalla del programa Aseprite*

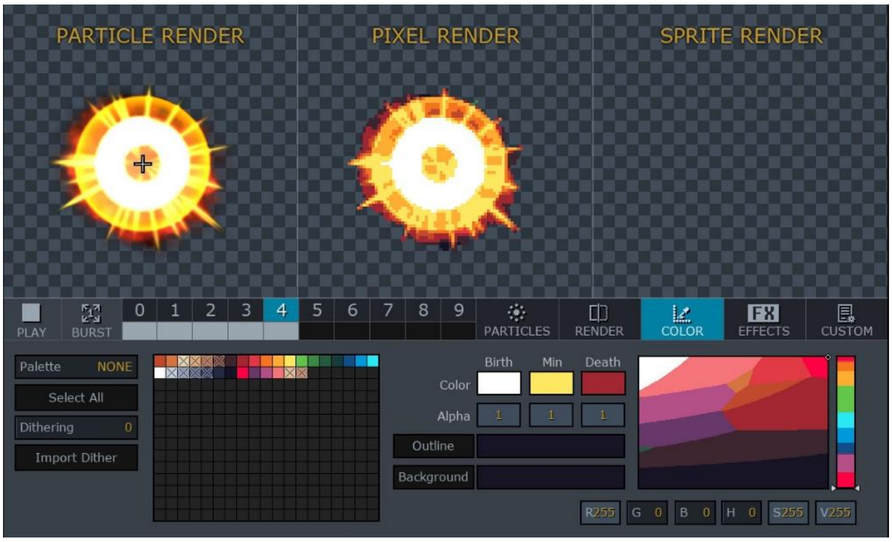


Figura 5.2: *Captura de pantalla del programa Pixel FX Designer*

	Aseprite	Pixel FX Designer
Estructura clàssica dels programes d'edició digital	X	
Paleta de colors disponible en qualsevol situació	X	
Espai per mostrar tots els paràmetres d'edició amb claredat		X
Accés fàcil a totes les eines del programa		X
Sistema de capes visible en tot moment		X
Més d'una finestra de treball	X	X

Taula 5.2: *Comparativa de les interfícies d'Aseprite i Pixel FX Designer*

Els sis punts destacats s'han tingut en compte per la creació de la interfície d'aquest treball per les següents raons:

- **Estructura clàssica dels programes d'edició digital**:
  Els usuaris estan acostumats a distribucions similars a les que presenta *Aseprite*, caracteritzades per una separació entre les paletes de color, les eines disponibles, les capes i un espai on treballar amb els paràmetres pels quals ha estat dissenyat el programa. Basant l'asset d'aquest TFG en aquesta estructura farà que els artistes es sentin còmodes amb la experiència oferta.

- **Paleta de colors disponible en qualsevol situació**:
  Una característica que no ofereix *Pixel FX Designer* és la possibilitat de canviar els colors del efectes creats en qualsevol punt del disseny, havent dedicat una secció a aquesta funcionalitat. En aquest cas, per afegir una secció d'edició de colors a l'abast de l'usuari en qualsevol moment, podent modificar valors d'altres funcionalitats a l'hora que s'experimenta amb diferents paletes.

- **Espai per mostrar tots els paràmetres d'edició amb claredat**:
  Un dels punts forts de *Pixel FX Designer* és que els paràmetres de cada funcionalitat compten amb un espai ampli de treball, el qual s'ha tingut en compte en el disseny d'aquest asset.

- **Accés fàcil a totes les eines del programa:**
  Mentre que en *Aseprite* hi ha una gran quantitat de funcionalitats amagades en diferents submenús, en *Pixel FX Designer* tots estan a la vista de l'usuari. Per oferir una experiència més amigable s'ha optat per simplificar i organitzar les funcionalitats disponibles com en aquest segon cas.

- **Sistema de capes visible en tot moment**:
  Ja que cada capa comptarà amb els seus paràmetres individuals és necessari que sempre estiguin a la vista de l'usuari, per aquesta raó s'ha optat per deixar un espai únicament dedicat a aquesta funcionalitat, amb la possibilitat d'ocultar capes amb un sol clic.

- **Més d'una finestra de treball**:
  El funcionament de l'asset, com s'ha explicat amb anterioritat, es basa en la llibreria de partícules de Unity, pel qual s'optarà per una estructura similar a la de *Pixel FX Designer* i es mostraran en una finestra la partícula en el seu estat original i en l'altra la mateixa partícula amb el filtre *pixel art* aplicat.

Amb les característiques del disseny de la interfície decidides, en el següent punt es passarà a mostrar els esquemes a partir dels quals es crearà la UI de l'asset.

5.1.3.2 Esquemes del Disseny de la UI

Esquema General

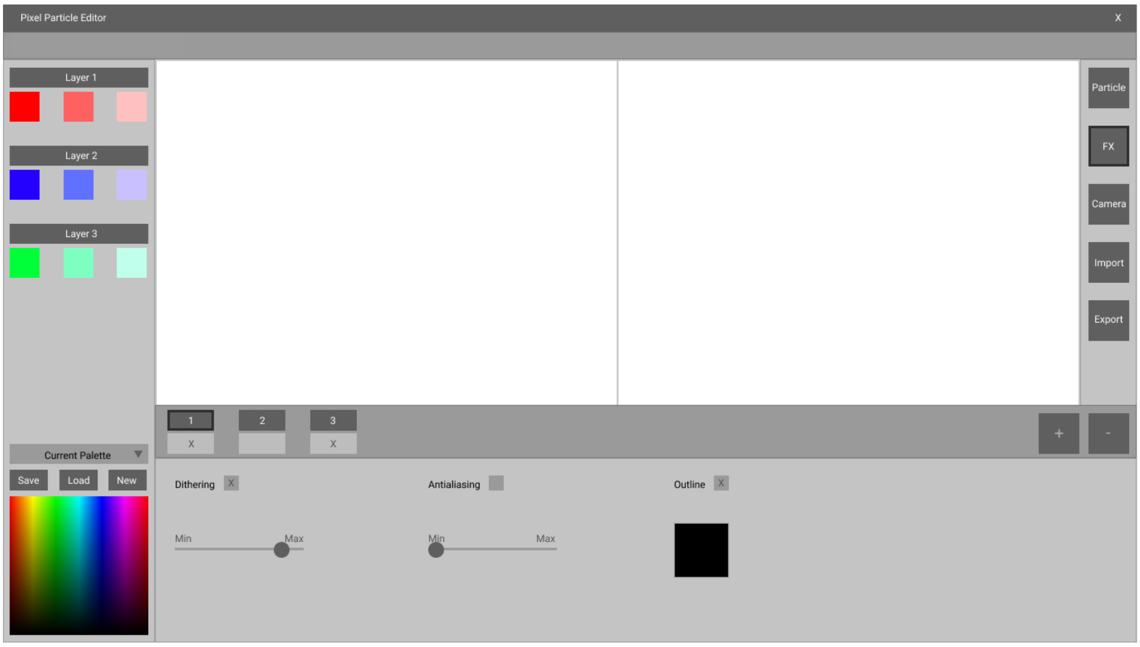


Figura 5.3: *Esquema General de la UI de l'asset*

En aquest esquema general es poden observar les grans parts en les que es dividirà la UI:

- **Colors i paletes:** Columna esquerra de l'esquema.
- **Funcionalitats:** Columna dreta de l'esquema.
- **Finestres de treball:** Quadrats centrals. Per defecte la finestra de l'esquerra mostrarà la partícula en estat original i la de la dreta amb el filtre *pixel art*.
- **Capes:** Barra central de l'esquema.
- **Paràmetres de la funcionalitat activa:** Barra inferior de l'esquema. El seu contingut canviarà segons la funcionalitat escollida en aquell moment.

Colors i Paletes

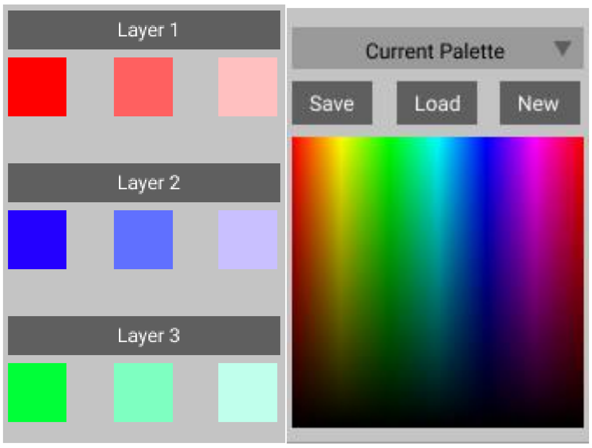


Figura 5.4: Esquema de l'apartat de Colors i Paletes de la UI de l'asset

En aquesta figura s'observen els elements que permetran a l'usuari canviar els colors dels seus *FX*.

- A la part superior de la columna (representada a la part esquerra de la figura 5.4) trobem els colors amb els que compta cada capa de partícules, els quals es podran modificar fent un clic i modificant el color a la part inferior de la columna.

- El primer color determina el color mostrat al naixement de la partícula, el color intermedi serà el que es mostri a la meitat de vida de la partícula i l'últim color serà el que tindrà quan aquesta desaparegui.

- La part inferior de la columna compta amb una llista desplegable de les paletes guardades, amb botons de guardar, carregar i creació d'una nova paleta. A baix trobem la paleta a partir de la qual es modificaran els colors de les creacions de l'usuari.

Funcionalitats

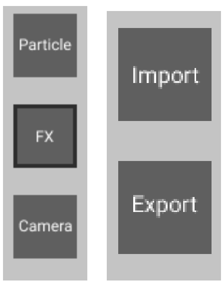


Figura 5.5: Esquema de l'apartat de Funcionalitats de la UI de l'asset

Aquí es pot veure la distribució de les funcionalitats del programa. La funcionalitat escollida estarà destacada en tot moment, com s'observa a la part esquerra d'aquesta figura.

La distribució ha sigut determinada segons la temàtica de cada funcionalitat, com s'explica a l'apartat de *back-end.*

Capes

Figura 5.6: *Esquema de l'apartat de Capes de la UI de l'asset*

A la part dreta d'aquesta barra es veuran totes les capes de la partícula amb la que s'està treballant.

La capa activa es veurà destacada com es veu a la part esquerra de la figura, sent la capa 1 la que està activa en aquest cas. Els quadres inferiors serviran per activat o desactivar la visibilitat de la capa en aquell moment, valor el qual es podrà canviar amb un sol clic.

La part dreta de la barra comptarà amb dos botons per crear o eliminar capes.

Paràmetres

Durant la fase de disseny de les funcionalitats es van plantejar diferents alternatives pel que fa a l'input de l'usuari. D'una banda, era interessant comptar amb totes les opcions oferides per la llibreria de partícules de Unity, però per l'altra es va considerar que si s'afegien totes aquestes funcionalitats no es tindria en compte l'objectiu d'aconseguir una bona experiència per a usuaris poc experimentats amb la creació de partícules.

Amb l'addició dels paràmetres d'importació es va solucionar la falta d'opcions per a usuaris experts, pel qual a l'hora de dissenyar la UI un dels punts clau ha sigut reduïr el màxim possible la barrera d'entrada a usuaris novells. Per aconseguir-ho s'ha optat per substituir la majoria dels valors originals que es troba a l'input per un apropament al que es coneix com *skeumorfisme*[12], sempre dins del que pot oferir la llibreria de l'editor de Unity.

En defensa a aquest tipus de disseny Alan Sien Wei Hshie, enginyer i dissenyador de UI d'Apple, va dir el següent fent referència a com els usuaris d'iPhone saben encendre el llum del seu mòbil perque està representat igual que a la vida real:

> *"Mentre que els usuaris poden saber com encendre un interruptor de llum, en algun moment van necessitar aprendre com es feia. Sempre hi haurà usuaris nous a una tecnologia o aparell, i els interruptors són molt més ubics que els iPhone."*

[12] L'*skeumorfisme* és una corrent de disseny popularitzada per l'empresa Apple en la qual es mostren elements virtuals simulant les seves contraparts reals, en favor de familiaritzar als usuaris amb les funcionalitats oferides pel dispositiu.

Si bé és complicat relacionar la creació de partícules per a un videojoc a quelcom tangible a la vida real, amb aquesta interfície s'ha intentat simplificar la interfície original, la qual compta amb una gran quantitat de menús i variables. Afegint *sliders*[13] a la gran majoria de paràmetres s'aconsegueix una experiència molt més intuitiva per a qualsevol usuari.

En les figures següents es poden veure els dissenys de cada quadre de paràmetres, els quals s'han basat en l'input definit a l'apartat de *back-end*.

- **Edició de Partícules**

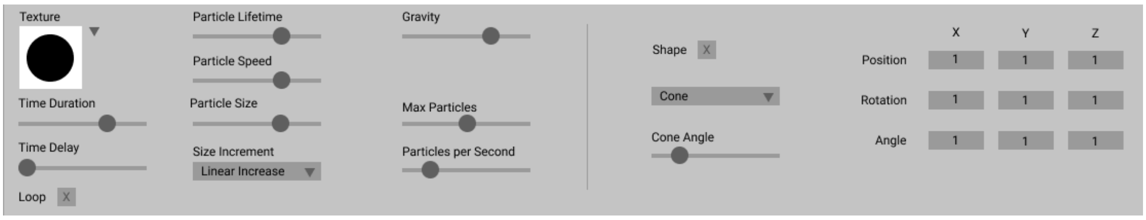


Figura 5.7: Esquema dels paràmetres d'edició de Partícules dins la UI de l'asset

- **Paràmetres d'Efectes Especials**

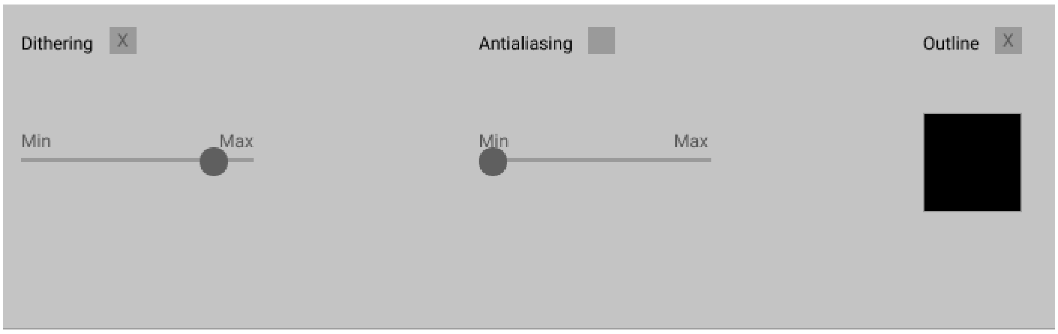


Figura 5.8: Esquema dels paràmetres d'Efectes Especials dins la UI de l'asset

- **Paràmetres de Càmera**

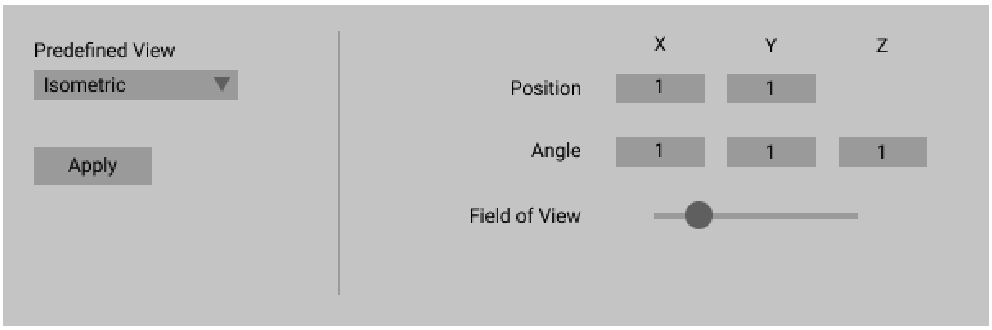


Figura 5.9: Esquema dels paràmetres de Càmera dins la UI de l'asset

[13] Un *slider* és un element de UI format per una barra i un botó el qual es pot lliscar per tot el recorregut d'aquesta barra. En aquesta interfície la posició del botó del *slider* determinarà valors numèrics.

- **Paràmetres d'Importació**

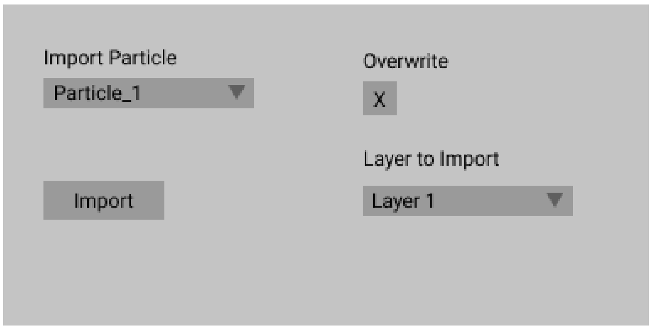


Figura 5.10: *Esquema dels paràmetres d'Importació dins la UI de l'asset*

- **Paràmetres d'Exportació**

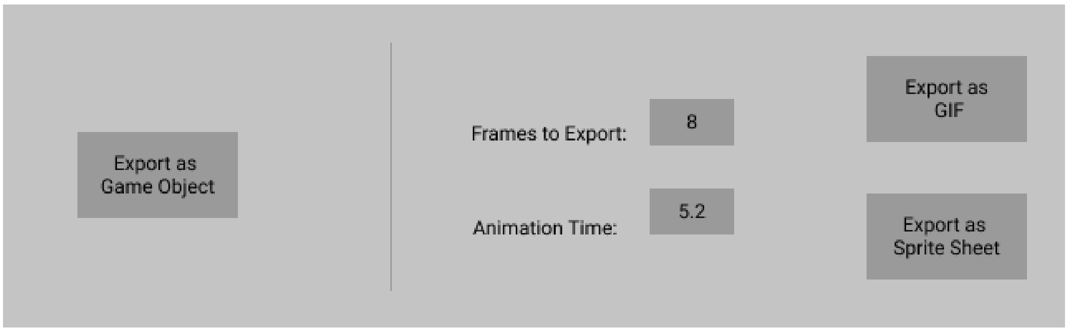


Figura 5.11: *Esquema dels paràmetres d'Exportació dins la UI de l'asset*

5.2 Producció

5.2.1 Implementació de la UI

A l'hora de decidir com implementar la UI de l'asset es van tenir en compte dues opcions. Per una banda està el sistema de Canvas de Unity, el qual compta amb una gran quantitat d'opcions fàcilment personalitzable a l'hora de realitzar UI i, a més, és la llibreria de Unity que més s'ha estudiat al llarg del grau pel que respecta a l'àmbit de les interfícies. Tot i aquests avantatges, té un punt negatiu que obliga a fer servir la següent alternativa. Aquest punt negatiu és que el Canvas està totalment enfocat a la creació de UI *in-game*[14], mentre que la segona opció s'enfoca en el que necessita aquest desenvolupament: la creació d'interfície personalitzada per a l'editor de Unity.

Aquesta alternativa és la llibreria *Editor* de Unity, la qual compta amb una gran quantitat de funcionalitats que veurem a continuació.

5.2.1.1 Inicialització de la UI

Per començar és necessari crear la finestra a partir de la qual es començarà a treballar en la interfície de l'asset. La llibreria *Editor* es pot destacar el següent codi per poder inicialitzar una finestra de treball al motor:

```
using UnityEditor;
```

Per poder tenir accés a les funcionalitats de l'Editor és necessari fer servir aquesta línia al principi de cada codi de UI.

```
public class PixelParticleEditor_UI : EditorWindow
```

De la mateixa manera, per fer saber al compilador que el codi es tracta d'una finestra de UI s'ha d'especificar al final de la declaració de la classe que es tracta d'un EditorWindow. En la majoria dels casos substituiria la declaració de MonoBehaviour de les classes de Unity.

```
[MenuItem("Window/Pixel Particle Editor")]
```

Amb aquesta línia s'especifica la localització de la finestra creada. En aquest cas es situa en l'apartat Window i portarà el nom de Pixel Particle Editor.

[14] El terme *in*-game es fa servir en la indústria del videojoc per fer referència a qualsevol cosa que aparegui exclusivament dins d'un videojoc. En aquest cas s'ha fet referència a que la interfície de la llibreria de Canvas només afecta a la part interna del joc i no pot modificar la interfície de l'editor.

```
static void Init()
    {
        EditorWindow editorWindow = GetWindow(typeof(PixelParticleEditor_UI));
    }
```

Aquesta porció de codi serà sempre necessària per inicialitzar la finestra. Sempre dins de Init(), s'ha de crear un EditorWindow i assignar-lo amb la funció GetWindow, la qual ha de tenir el nom de la classe del codi (PixelParticleEditor_UI en aquest cas).

```
private void OnGUI()
    {
        GUILayout.Label("Hello World", EditorStyles.boldLabel);
    }
```

Dins de la funció OnGUI() s'escriurà tot el codi que donarà forma a la part visual de la UI. En aquest cas s'ha afegit una *label*, és a dir, un fragment de text, el qual forma la ja clàssica frase "Hello World".

Amb tot aquest codi s'ha aconseguit el següent resultat:

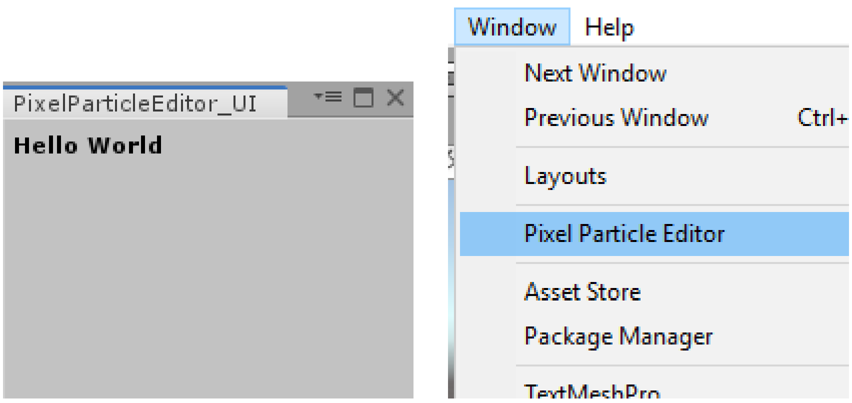


Figura 5.12: *Finestra de Hello World dins del motor Unity*

5.2.1.2 Distribució de les seccions

Ara que ja es té una finestra en funcionament és el moment de dividir els espais de treball per poder afegir cadascuna de les seccions dissenyades. Les següents línies de codi seran las que facilitaran aquest procés:

```
EditorGUILayout.BeginHorizontal();
//El contingut afegit aquí es distribuirà de manera horitzontal
EditorGUILayout.EndHorizontal();

EditorGUILayout.BeginVertical();
//El contingut afegit aquí es distribuirà de manera vertical
EditorGUILayout.EndVertical();
```

En cas de voler una distribució d'elements horitzontal s'han d'afegir les funcions BeginHorizontal() i EndHorizontal() que trobem a la classe EditorGUILayout, fent que qualsevol element entre aquestes dues línies es distribueixi horitzontalment. Si, d'altra banda, es desitja una distribució vertical el funcionament és el mateix, però amb les funcions BeginVertical() i EndVertical().

```
Vector2 scroll_test;

scroll_test = EditorGUILayout.BeginScrollView(scroll_test, GUILayout.Width(350),
GUILayout.Height(350));
//El contingut afegit aquí es distribuirà dins de l'espai assignat a la línia
anterior
EditorGUILayout.EndScrollView();
```

La classe BeginScrollView facilita la creació d'espais delimitats. Aquesta classe compta amb un atribut Vector2, el qual guarda els valors de la posició de la pantalla, i dos valors Width i Height, els quals determinen la mida de la finestra (en aquest cas 350 píxels cadascun dels valors). El contingut s'ha d'afegir entre les línies de BeginScrollView() i EndScrollView(), igual que en el cas anterior.

Afegint *labels* per especificar cada zona de la finestra, el resultat aconseguit és el següent:

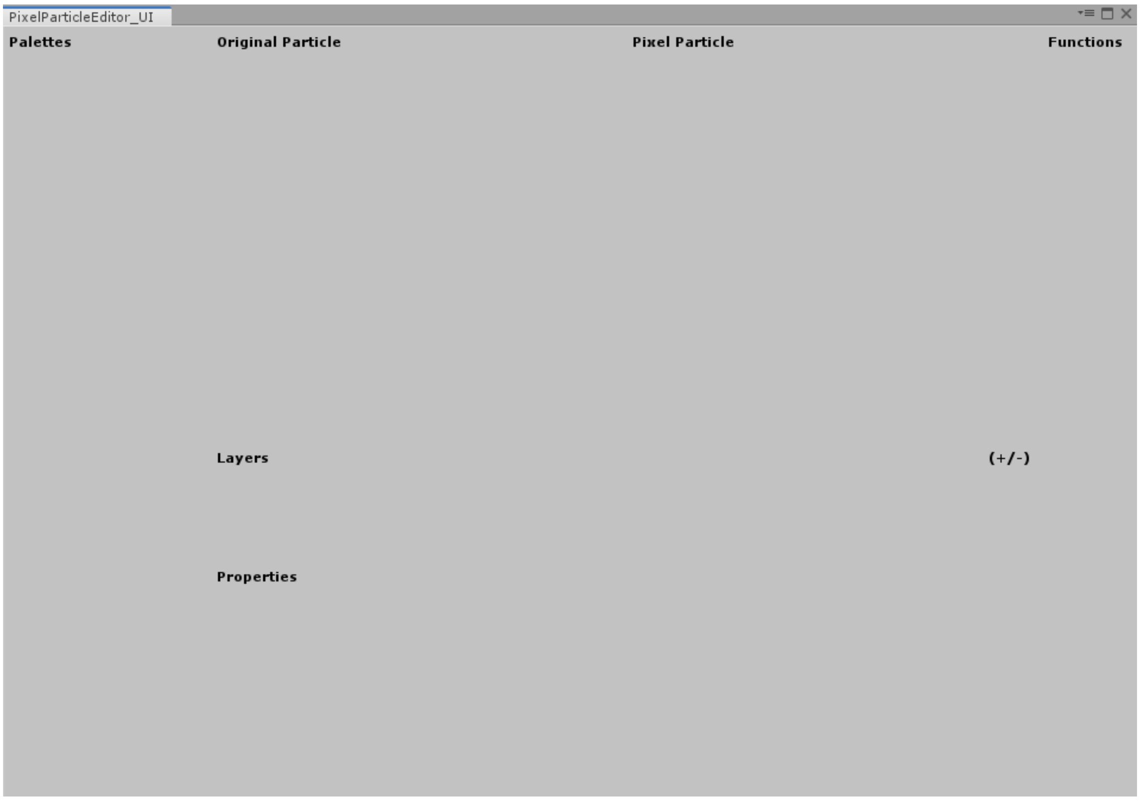


Figura 5.13: *Finestra de l'asset amb els espais distribuïts*

5.2.1.3 Elements de Input

Ara que els elements de pantalla estan distribuïts la finestra queda preparada per afegir els elements d'input. Aquests són els elements que seran útils per la creació d'aquesta UI:

```
Color color_test;

color_test = EditorGUILayout.ColorField(color_test);
```

La funció ColorField de la classe EditorGUILayout serveix per introduir un quadre de modificació de color. És necessari assignar una variable Color, en aquest cas color_test, la qual es podrà modificar amb l'input de l'usuari. Aquest element servirà per crear l'apartat de paletes.

```
bool checkbox_test;

checkbox_test = GUILayout.Toggle(checkbox_test,"Checkbox Test");
```

La funció Toggle serveix per afegir una *checkbox,* és a dir, una casella que pot estar activada i desactivada. Té un funcionament molt semblant a l'anterior, amb l'addició d'un atribut de text que determina el text que acompanyarà la *checkbox.* Pel funcionament d'aquest element és necessari assignar el seu valor a una variable *bool* (checkbox_test en aquest cas).

```
bool button_test;

if (GUILayout.Button("Button Test", GUILayout.Width(30), GUILayout.Height(30)))
            button_test = !button_test;
```

La funció Button de la classe GUILayout afegeix un botó a la interfície, el qual activa el contingut de l'*if* cada vegada que és polsat. Aquest element compta amb variables Width i Height que determinen la seva mida.

```
float slider_test = 0.0f;

slider_test = GUILayout.HorizontalSlider(slider_test, 0, 100);
```

Per afegir un *slider* a la UI es fa servir la funció HorizontalSlider de la classe GUILayout, al qual se l'assigna una variable de tipus float a la qual modifica el seu valor segons la posició del *slider*. Als seus atributs compta amb dos valors que determinen els valors mínims i màxims del recorregut del botó.

```
Texture2D texture_test;

texture_test = (Texture2D)EditorGUILayout.ObjectField(texture_test,
typeof(Texture2D), false);
```

La funció ObjectField de la classe EditorGUILayout és molt útil, ja que permet a l'usuari afegir un *Game Object* del tipus que es determini al codi. En aquest cas l'element a afegir serà de tipus *Texture2D* (un arxiu de imatge). Als seus atributs s'assigna la variable a modificar, el tipus d'element i un *bool* que determina si es poden afegir o no elements de l'escena de joc.

Una vegada s'han afegit tots els elements en pantalla, així queda la pantalla de l'*asset*.

***Figura 5.14:** Finestra de l'asset amb els elements d'input afegits*

5.2.1.3 Renderitzat de les partícules

Per terminar amb les funcionalitats bàsiques de la UI és necessari afegir el renderitzat de partícules. El següent codi facilitarà la inclusió d'aquesta característica:

```
Camera camera;
RenderTexture renderTexture;

public void Update()
    {
        if (camera != null)
        {
            camera.targetTexture = renderTexture;
            camera.Render();
            camera.targetTexture = null;
        }
    }
```

Per aconseguir que les partícules es renderitzin s'afegeix una càmera que passarà tot el que capturi a una RenderTexture, una imatge que s'actualitzarà amb les dades que li passarà la Camera.

A la funció Update(), la qual es crida a cada *frame*, s'assignarà la variable renderTexture a l'atribut targetTexture. Després es cridarà la funció Render() de Camera per passar les dades d'imatge a renderTexture (imatge que es pintarà en pantalla amb el codi que ve a continuació).

```
private void OnGUI()
{
	GUI.DrawTexture(new Rect(0.0f, 0.0f, position.width, position.height),
renderTexture);
}
```

Per poder representar els elements en la interfície es crida a la funció DrawTexture, que serveix per pintar imatges en pantalla. Com la variable renderTexture s'actualitza cada *frame* amb la imatges capturades per la càmera.

A partir del codi explicat, s'aconsegueix la primera versió de la interfície de l'*asset*, la qual s'anirà actualitzant a mesura que s'afegeixin les funcionalitats pendents per desenvolupar:

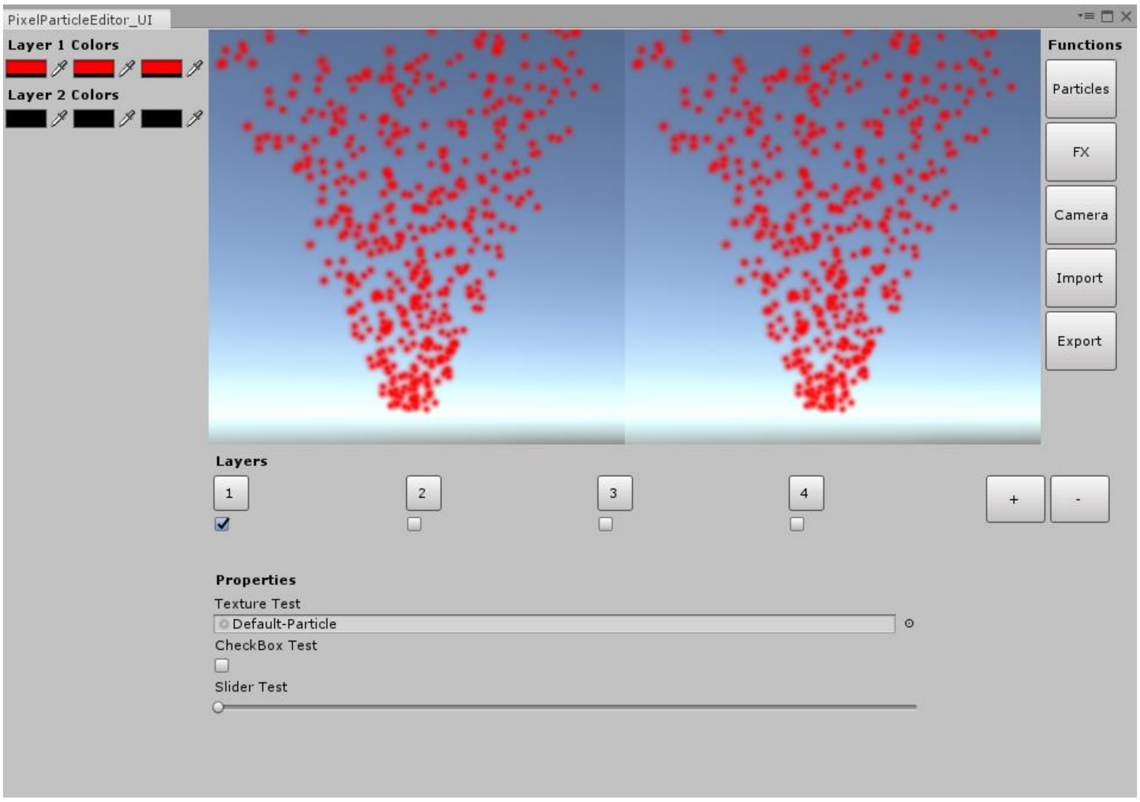


Figura 5.15: *Finestra de l'asset amb el renderitzat de partícules actiu*

5.2.2 Renderitzat Pixel Art i Shader Base

5.2.2.1 Resolució Reduïda

Per començar en la creació del renderitzat estil *pixel art* s'ha tingut en compte que el pintat d'imatges de la UI es realitza amb un RenderTexture, així que el primer pas ha sigut crear una RenderTexture la qual es modificarà posteriorment per donar efecte *pixel art*.

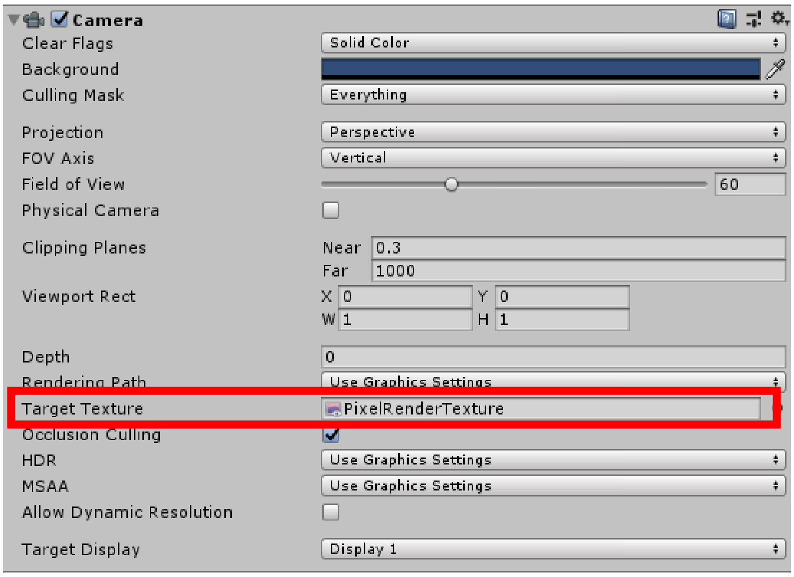


Figura 5.16: *Paràmetres de càmera de Unity amb Target Texture destacat*

Una vegada afegida la RenderTexture a la càmera es modifica la resolució de renderitzat per tal d'aconseguir l'efecte *pixel art* desitjat. La següent tasca serà programar el *shader* per aconseguir una major qualitat pel que fa al *pixel art* en el renderitzat de les partícules. Per tal de que les cantonades quedin definides s'aplicarà el filtre de tipus Point, tal com es mostra a la següent figura.

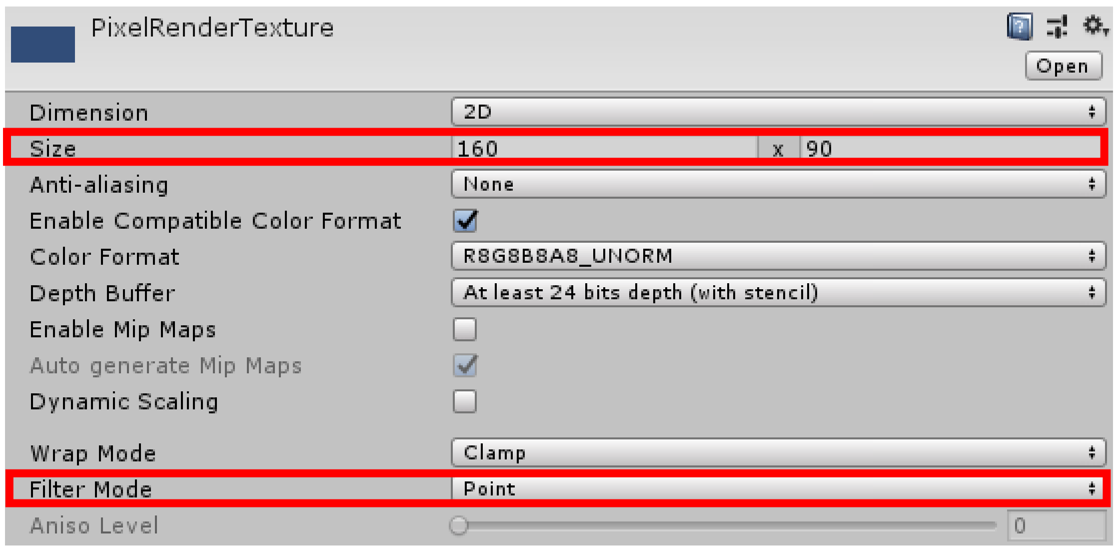


Figura 5.17: *Paràmetres de RenderTexture*

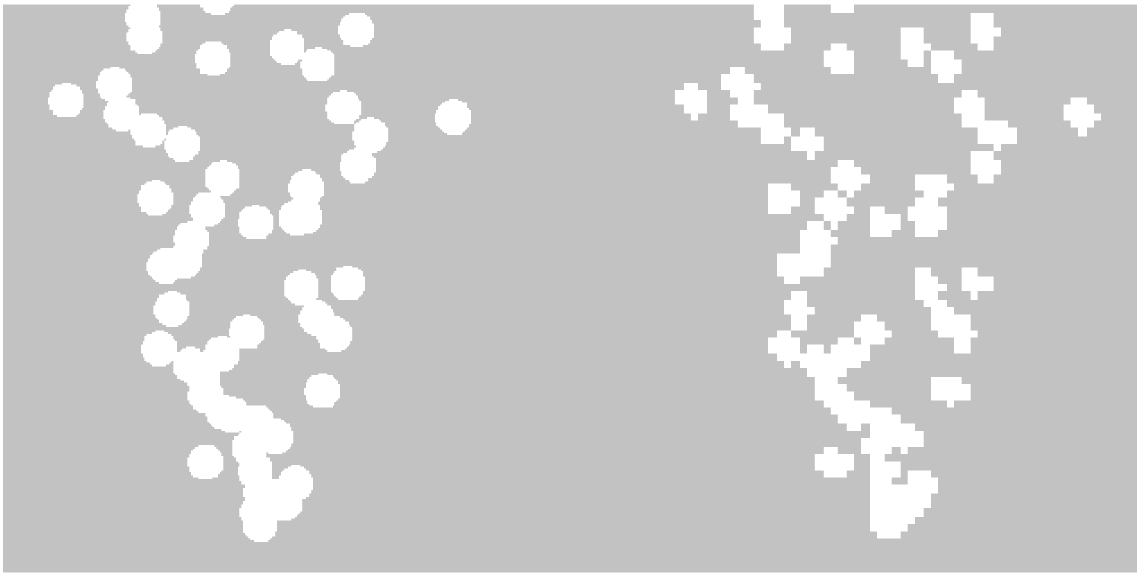


Figura 5.18: *Comparativa entre renderitzat normal i renderitzat amb resolució reduïda*

5.2.2.2 Shader Base amb Color Blending

Fins aquest punt s'ha fet servir de manera provisional un *shader* de tipus Unlit inclòs al motor Unity, però aquest no permet modificar el color de les textures segons els paràmetres del sistema de partícules. Per aquesta raó s'ha realitzat un *shader* Unlit personalitzat, el qual comptarà amb els següents paràmetres a l'apartat "Properties" del mateix:

```
Properties
	{
		_MainTex("Texture", 2D) = "white" {}
		_Cutoff("Alpha cutoff", Range(0,1)) = 0.5
	}
```

_MainTex ve assignat per defecte, sent la textura que acompanyarà les partícules del *shader.* _Cutoff és un valor afegit per permetre retallar la transparència de la textura, evitant soroll a la imatge i així deixant-la més neta. Aquests valors s'han d'aplicar al codi intern del *shader* com es veu a continuació:

```
struct appdata
		{
			float4 vertex : POSITION;
			float2 uv : TEXCOORD0;
			float4 color : COLOR;
		};
```

En primer lloc s'ha de declarar la variable de color, la qual s'afegirà al struct que enviarà les dades al programa de vèrtex que veurem després. La variable ha de ser de tipus float4, la qual inclourà els valors R, G, B i A del color.

```
struct v2f
            {
                  float2 uv : TEXCOORD0;
                  UNITY_FOG_COORDS(1)
                  float4 vertex : SV_POSITION;
                  float4 color : COLOR;
            };
```

De la mateixa manera, al struct v2f del *shader* s'ha de declarar una variable del mateix tipus per enviar el color al programa de fragment que es mostrarà a continuació.

```
v2f vert(appdata v)
            {
                  v2f o;
                  o.vertex = UnityObjectToClipPos(v.vertex);
                  o.uv = TRANSFORM_TEX(v.uv, _MainTex);
                  UNITY_TRANSFER_FOG(o,o.vertex);
                  o.color = v.color;
                  return o;
            }
```

Dins del programa de vèrtex del *shader* s'assigna el valor color de la variable a retornar ("o" en aquest cas) pel valor de color de la variable rebuda ("v" en aquest cas).

```
fixed4 frag(v2f i) : SV_Target
            {
                  // sample the texture
                  fixed4 col = tex2D(_MainTex, i.uv) * i.color;
                  clip(col.a - _Cutoff);
                  // apply fog
                  UNITY_APPLY_FOG(i.fogCoord, col);
                  return col;
            }
```

Al programa de fragment s'ha de multiplicar el valor de color del programa de vèrtex rebut ("i" en aquest cas) tal i com es veu al codi. També s'afegirà la funció clip per retallar la capa de transparència segons el valor decidit al _Cutoff definit al principi del codi.

Per acabar el *shader* s'ha de definir el tipus de render, el qual determinarà si les textures es renderitzen de manera opaca o amb transparència, de la mateixa manera que el valor de la cua de renderitzat, la qual determina el tipus de pintat entre textures superposades:

```
SubShader
	{
		Tags{ "RenderType" = "Transparent" "Queue" = "Transparent" }

		Zwrite Off
```

Fent servir la etiqueta "Transparent" pels valors de RenderType (tipus de renderitzat) i Queue (cúa de renderitzat) s'aconseguirà que la transparència de la partícula funcioni correctament tant en partícules independents com a partícules superposades. El Zwrite decideix si el renderitzat té en compte la capa de profunditat. Ja que s'està treballant amb partícules 2D semitransparents l'ideal és apagar aquesta funcionalitat.

Pel que fa al color, s'han d'afegir un seguit de valors al SubShader, just després de la línia de codi de ZWrite Off mostrada anteriorment:

`Blend SrcAlpha OneMinusSrcAlpha`
`Cull Back`
`Pass`
	`Blend One One`
	`Cull Back`
	`Pass`
		`BlendOp RevSub`
		`Blend One One`
		`Cull Back`
		`Pass`

Els tipus de renderitzat de color disponibles són molt variats, pel qual s'han realitzat tres *shaders* diferents, cadascun amb un tipus de pintat de color (o Color Blending) diferent.

El primer compta amb un *Blending* normal, el qual aplicarà totalment el color determinat per l'usuari, sense cap tipus de transparència.

El segon compta amb un *Blending* de tipus additiu, el qual pintarà els colors amb transparència i sumarà els valors dels colors que es superposin.

El tercer compta amb un *Blending* substractiu, el qual eliminarà totalment el color de les partícules superposades.

A la següent figura es poden observar exemples dels tres tipus de Color Blending en total funcionament dins de l'*asset*.

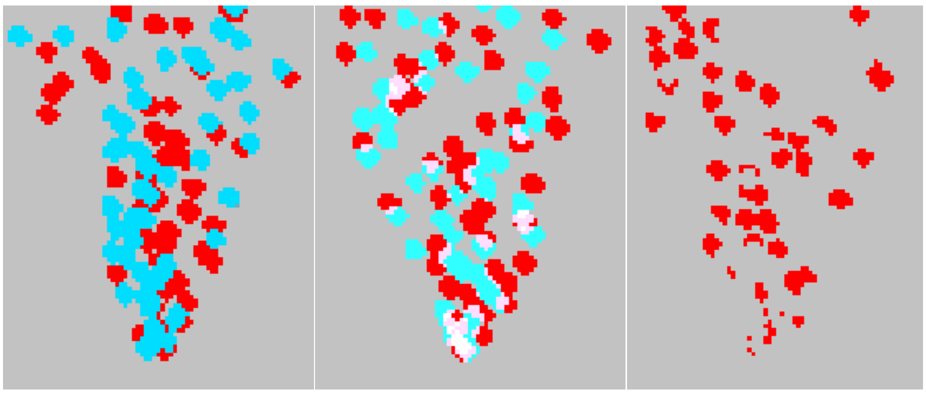


Figura 5.19: *Comparativa dels Color Blending disponibles*

5.2.3 Opcions d'exportació

5.2.3.1 Exportació de GameObject

La primera opció d'exportació afegida serà la d'exportar les partícules com a Game Object, per tal de poder fer-les servir en qualsevol projecte de Unity. Amb el següent codi es poden guardar fàcilment les partícules creades a l'editor en aquest format:

```
string file_name;

Particle particle

ParticleSystemRenderer particle_renderer;
particle_renderer = particle.GetComponent<ParticleSystemRenderer>();

Material export_material = new Material(particle_renderer.material);

AssetDatabase.CreateAsset(export_material, "Assets/" + file_name +
"_Material.mat");
```

En primera instància, abans d'exportar la partícula es desarà el material d'aquesta. Amb la funció CreateAsset es pot aconseguir aquest objectiu. En aquest cas el material s'exportarà a la carpeta principal d'Assets amb el nom assignat a file_name seguit de "_Material".

Cap destacar que s'ha creat un nou material idèntic al de la partícula per així desvincular-lo del que està en funcionament dins l'editor i evitar errors en cas de modificar el material exportat fora de l'entorn de treball de l'*asset*.

```
GameObject to_export = Instantiate(particle.gameObject, new Vector3(0, -90, 0),
Quaternion.identity);

ParticleSystemRenderer export_renderer =
to_export.GetComponent<ParticleSystemRenderer>();

export_renderer.material = export_material;
```

De la mateixa manera que amb el material, s'ha creat una partícula idèntica a la que es vol exportar amb la funció Instantiate, fent que l'arxiu desat estigui desvinculat dels paràmetres de *l'asset*.

```
PrefabUtility.SaveAsPrefabAsset(to_export, "Assets/" + file_name + ".prefab");

DestroyImmediate(to_export);
```

Per acabar s'exporta la partícula en format Prefab a la mateixa carpeta que el material amb la funció SaveAsPrefabAsset dins de PrefabUtility. Una vegada s'hagi desat l'arxiu s'eliminarà de l'escena amb DestroyImmediate.

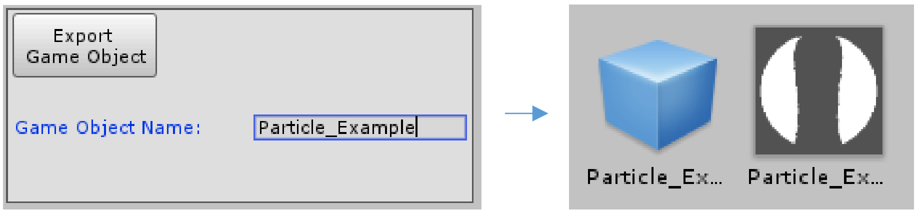


Figura 5.20: *Exemple de l'exportació de Game Object en funcionament*

5.2.3.2 Exportació de Sprite Sheet

En cas que algun usuari vulgui modificar el FX creat amb algun programa d'edició d'imatge serà necessari crear un arxiu d'imatge en primer lloc. Amb aquest *asset* es podrà exportar qualsevol partícula en format *Sprite Sheet* png, el qual es realitzarà de la següent manera:

```
if (GUILayout.Button("Export\n Sprite Sheet", GUILayout.Width(90),
GUILayout.Height(40)))
        {
            render_active = true;
            wait_one_frame = false;
            current_frame = 0;
        }
```

Ja que per realitzar aquesta tasca serà necessari realitzar un seguit de funcions dins la funció Update() del codi al polsar el botó de renderitzat s'activarà un bool que avisarà la funció esmentada per realitzar l'exportació.

```
float current_time = render_time / render_frames * current_frame;
particle.Simulate(current_time);
current_frame++;

if (wait_one_frame == false)
{
       wait_one_frame = true;
}

else
{
       SaveFrameToList();

       if (current_frame >= render_frames+1)
       {
              SaveSpritesheet();
              particle.Play();
              render_active = false;
       }
}
```

Aquest codi funciona dins de la funció Update() de la UI ja que ha de exportar *frame* per *frame* l'animació de la partícula creada. Tenint en compte que a la interfície l'usuari donarà els valors de render_time i render_frames, en primer lloc es calcularà el temps exacte de simulació del *frame* a guardar, el qual es farà servir per simular el *frame* exacte amb la funció Simulate(time).

Ja que la simulació es realitza al final de l'Update, el primer *frame* s'esperarà amb el bool wait_one_frame i així poder exportar correctament l'animació.

Una vegada estigui tot llest per desar les imatges a una llista de Texture2D amb SaveFrameToList(), la qual es veurà a continuació.

Quan estiguin tots els *frames* llestos es passarà a desar el *Sprite Sheet* amb SaveSpritesheet(), el qual es veurà al final d'aquest apartat.

```
List<Texture2D> rendered_textures = new List<Texture2D>();

public void SaveFrameToList()
    {
        Texture2D new_texture = CreateTexture2D(renderTexture);
        rendered_textures.Add(new_texture);
    }
```

Per desar les partícules renderitzades a la UI en format textura es fa servir la funció CreateTexture2D afegint com argument el renderTexture que s'ha fet servir per pintar en estil *pixel art* els FX. Una vegada creada la textura s'afegeix aquesta a una llista de textures prèviament declarada.

```
public void SaveSpritesheet()
    {
        Texture2D texture = new Texture2D(frame_size*render_frames, frame_size);

        for (int i = 0; i < render_frames; i++)
        {
            for (int x = 0; x < frame_size; x++)
            {
                for (int y = 0; y < frame_size; y++)
                {
                    Color to_render = rendered_textures[i].GetPixel(x, y);
                    texture.SetPixel(x + frame_size *i, y, to_render);
                }
            }
        }

        byte[] bytes = texture.EncodeToPNG();

        string path = Application.dataPath + file_name + ".png";
        System.IO.File.WriteAllBytes(path, bytes);

        rendered_textures.Clear();
    }
```

Per exportar l'arxiu final de *Sprite Sheet* en primer lloc s'ha de crear una nova Texture2D de la mida que permeti incloure tots els *frames* desitjats. En aquest exemple es crearà un *Sprite Sheet* horitzontal on, d'esquerra a dreta, s'exportarà cada *frame* de l'animació de la partícula.

Gràcies a les funcions GetPixel i SetPixel aquest codi recorrerà tota la matriu de píxels de les Texture2D desades prèviament a la llista i les afegirà al seu lloc corresponent en la textura a exportar.

Una vegada està llesta la textura es crearà un PNG amb la funció EncodeToPNG de Texture2D i es desarà l'arxiu amb la funció WriteAllBytes(), tal i com es mostra al codi. Per acabar es neteja la llista de textures per tal de poder realitzar una nova exportació sense problemes.

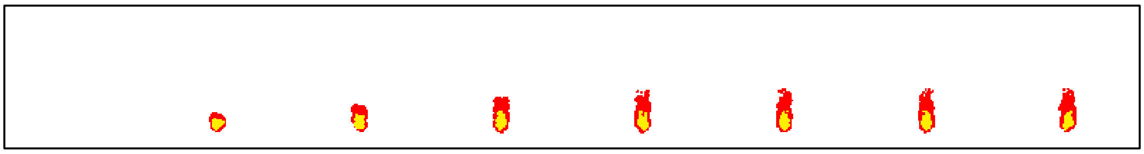


Figura 5.21: *Exemple de Sprite Sheet exportat amb l'asset*

5.2.4 Paletes de color personalitzables

5.2.4.1 Script d'aplicació de shader a càmera

Ara que totes les funcionalitats d'exportació estan implementades és el torn d'implementar les paletes de color limitades. En aquest cas s'implementarà un límit de quatre colors a la imatge renderitzada per la càmera, a la qual se l'aplicarà un *shader.*

En primer lloc s'ha de crear un *script* simple que s'aplicarà a la càmera i determinarà el *shader* que es farà servir:

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

[ExecuteInEditMode]
public class CameraPalette : MonoBehaviour
{
    public Material palette_material;

    private void OnRenderImage(RenderTexture src, RenderTexture dst)
    {
        Graphics.Blit(src, dst, palette_material);
    }
}
```

Abans de declarar el *script* s'ha d'activar el mode d'execució a l'editor amb l'etiqueta [ExecuteInEditMode], ja que l'*asset* desenvolupat funcionarà només en el mode d'edició del motor.

L'única funcionalitat que tindrà aquest codi serà modificar la imatge generada per la càmera. Per fer-ho, dins de la funció OnRenderImage(), s'utilitzarà Graphics.Blit(src, dst, material), sent "src" el RenderTexture d'origen, "dst" el RenderTexture on s'aplicarà la imatge i "material" el material que inclourà el *shader* que es desenvoluparà a continuació.

5.2.4.2 Shader de limitació de colors

Mentre que el *shader* creat per les partícules era de tipus unlit el que farà servir la càmera és un d'efecte d'imatge (Image Effect Shader). Per començar s'han d'afegir dins de Properties els valors que l'usuari podrà modificar:

```
Properties
    {
              _MainTex("Texture", 2D) = "white" {}
              _Darkest("Darkest", color) = (0.0588235, 0.21961, 0.0588235)
              _Dark("Dark", color) = (0.188235, 0.38431, 0.188235)
              _Light("Light", color) = (0.545098, 0.6745098, 0.0588235)
              _Lightest("Lightest", color) = (0.607843, 0.7372549, 0.0588235)
    }
```

Aquest *shader* estarà limitat a quatre colors, els quals funcionaran segons la quantitat de lluminositat de la imatge renderitzada. Els quatre colors afegits estan dividits des del color més fosc fins el més clar.

```
sampler2D _MainTex;
float4 _Darkest, _Dark, _Light, _Lightest;

fixed4 frag (v2f i) : SV_Target
{
       ...
}
```

De la mateixa manera, també es declararan les variables abans d'entrar al programa de fragment del *shader*.

Figura 5.22: *Propietats del shader de paletes personalitzades a l'editor*

```
fixed4 frag (v2f i) : SV_Target
{
            float4 originalColor = tex2D(_MainTex, i.uv);

            float luma = dot(originalColor.rgb, float3(0.2126, 0.7152, 0.0722));
            float posterized = floor(luma * 4) / (4 - 1);

            ...
```

Dins del programa de fragment del *shader* s'obtindrà en primera instància el color original a renderitzar enviant la textura de la mateixa manera que es va fer al *shader* de partícules.

Després es passa a calcular el valor de lluminositat desaturant el color original amb el producte vectorial del color amb el vector de lluminositat[15] d'aquest.

Una vegada es coneix el valor de lluminositat s'obtindrà l'índex del color a utilitzar. Fent servir floor(x) s'obté el valor enter menor o igual a "x", pel qual es multiplicarà la lluminositat per la quantitat de colors a pintar (4 en aquest cas). Una vegada obtingut aquest valor es divideix per la quantitat de colors menys 1 per obtenir el valor desitjat. En aquesta situació es poden donar cinc valors: 0, 1/3, 2/3, 1 i 4/3. 0 serà el valor més fosc i 4/3 el més clar, compartint tonalitat amb 1.

```
            ...

            float lumaTimesThree = posterized * 3.0;

            float darkest = saturate(lumaTimesThree);
            float4 color = lerp(_Darkest, _Dark, darkest);

            float light = saturate(lumaTimesThree - 1.0);
            color = lerp(color, _Light, light);

            float lightest = saturate(lumaTimesThree - 2.0);
            color = lerp(color, _Lightest, lightest);

            color.a = originalColor.a;

            return color;
}
```

Per acabar es multiplicarà el valor per tres i s'assignarà a una nova variable. Aquest valor es farà servir amb la funció saturate, que funciona de la següent manera:

saturate(x) = max(0, min(1, x));

El valor que retorna és el màxim entre 0 i el mínim entre 1 i el valor de "x". Ja que tots els valors possibles són enters, hi ha dos resultats possibles: 0 i 1.

[15] El vector de lluminositat escollit és (0.2126, 0.7152, 0.0722). Aquest valor ha sigut extret del web http://www.brucelindbloom.com/index.html?WorkingSpaceInfo.html.

La següent funció que es fa servir a cada iteració és lerp(a, b, w). Aquesta funció fa una interpolació entre els valors "a" i "b" mitjançant el pes de "w". Ja que el valor de "w" en aquest cas serà 0 o 1 el que farà aquesta funció és escollir entre "a" i "b", retornant "a" quan "w" sigui igual a 0 i "b" quan sigui "w" sigui igual a 1.

Realitzant tres iteracions, restant 1 a cada iteració tal i com es veu al codi d'exemple, s'aconsegueix limitar la paleta als quatre colors definits per l'usuari.

Per poder eliminar el fons del RenderTexture s'iguala al final del codi el valor *alpha* del color original al color a retornar amb color.a = originalColor.a.

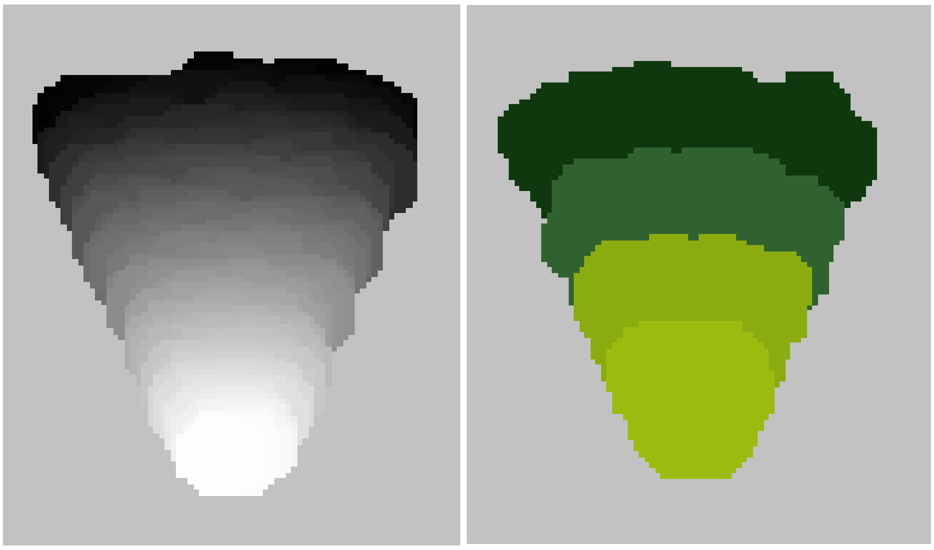


Figura 5.23: *Comparativa entre partícula amb render normal i amb el shader de paleta de colors*

5.2.5 Dithering

La tècnica de *dithering*, com s'ha explicat a l'apartat d'Estat de l'Art, es fa servir per afegir una transició suau entre tonalitats a paletes limitades. És per aquesta raó s'ha decidit que la funcionalitat de *dithering* s'aplicarà al *shader* de paletes explicat anteriorment.

```
Properties
	{
		_MainTex("Texture", 2D) = "white" {}
		_DitherTex("Dither Texture", 2D) = "" {}
		_Darkest("Darkest", color) = (0.0588235, 0.21961, 0.0588235)
		_Dark("Dark", color) = (0.188235, 0.38431, 0.188235)
		_Ligt("Light", color) = (0.545098, 0.6745098, 0.0588235)
		_Ligtest("Lightest", color) = (0.607843, 0.7372549, 0.0588235)
	}

	...

	sampler2D _MainTex;
	float2 _MainTex_TexelSize;

	sampler2D _DitherTex;
	float2 _DitherTex_TexelSize;

	float4 _Darkest, _Dark, _Ligt, _Ligtest;

	...
```

Com als casos anteriors s'ha de començar afegint els nous paràmetres a modificar per l'usuari. En aquest *shader* s'utilitzaran textures per aplicar l'efecte de *dithering*, així que com s'observa al codi d'exemple s'afegeix un nou paràmetre de tipus textura 2D.

Per la lògica d'aquest codi seran necessaris els valors de la mida de la textura principal i la de *dithering*, així que seran declarades juntament amb les textures.

```
fixed4 frag(v2f i) : SV_Target

	{
		float2 dither_uv = i.uv * _DitherTex_TexelSize;
		dither_uv /= _MainTex_TexelSize;
		float dither = tex2D(_DitherTex, dither_uv).a;

		...
```

Dins del programa de fragment el primer que es farà és buscar la posició de la textura de *dither* a la textura de render. Per realitzar-ho s'ha de multiplicar la posició de la textura original per la mida de la textura de *dithering*, i aquesta després s'ha de dividir per la mida de la textura original.

Amb aquesta posició podrem obtenir el valor de transparència de la textura de *dithering* en aquest punt exacte del RenderTexture, valor que es farà servir per aplicar el *dithering* amb el codi que hi ha a continuació.

```
...

float luma = dot(originalColor.rgb, float3(0.2126, 0.7152, 0.0722));
float posterized = floor(luma * 8) / (8 - 1);

float lumaTimesSeven = posterized * 7.0;

float darkest = saturate(lumaTimesSeven);
float4 color = lerp(_Darkest, _Dark, darkest);

float darkest2 = saturate(lumaTimesSeven - 1.0);
color = lerp(color, _Dark, darkest2);

float4 dither_color = (luma*3.0) < dither ? _Darkest : color;
color = lerp(color, dither_color, darkest);
```

En aquest *shader* la paleta s'ha multiplicat pel doble de colors per tal de poder realitzar els càlculs del *dithering* als nous punts d'intersecció de color.

Ara cada tonalitat tindrà dues comprovacions amb el mateix funcionament que l'explicat al punt anterior. Després d'aquestes comprovacions es realitza la comparació entre el valor de lluminositat corresponent i el del *dither*, com es veu a la part destacada del codi. En cas que el valor de la lluminositat sigui inferior al valor de *dither* s'aplica el color anterior de la paleta, sinó continuarà amb el valor actual. El codi resultant amb totes les iteracions de color seria el següent:

```
float darkest = saturate(lumaTimesSeven);
float4 color = lerp(_Darkest, _Dark, darkest);

float darkest2 = saturate(lumaTimesSeven - 1.0);
color = lerp(color, _Dark, darkest2);

float4 dither_color = (luma*3.0) < dither ? _Darkest : color;
color = lerp(color, dither_color, darkest);

float light = saturate(lumaTimesSeven - 2.0);
color = lerp(color, _Light, light);

float light2 = saturate(lumaTimesSeven - 3.0);
color = lerp(color, _Light, light2);

float4 dither_color2 = (luma*3.0-1.0) < dither ? _Dark : color;
color = lerp(color, dither_color2, light);

float lightest = saturate(lumaTimesSeven - 4.0);
color = lerp(color, _Lightest, lightest);

float lightest2 = saturate(lumaTimesSeven - 5.0);
color = lerp(color, _Lightest, lightest2);

float4 dither_color3 = (luma*3.0-2.0) < dither ? _Light : color;
color = lerp(color, dither_color3, lightest);

color.a = originalColor.a;

return color;
```

Segons el tipus de textura que es faci servir el patró de *dithering* resultant serà diferent. A continuació es poden veure exemples de *dithering* amb textures de 2x2, 4x4 i 8x8:

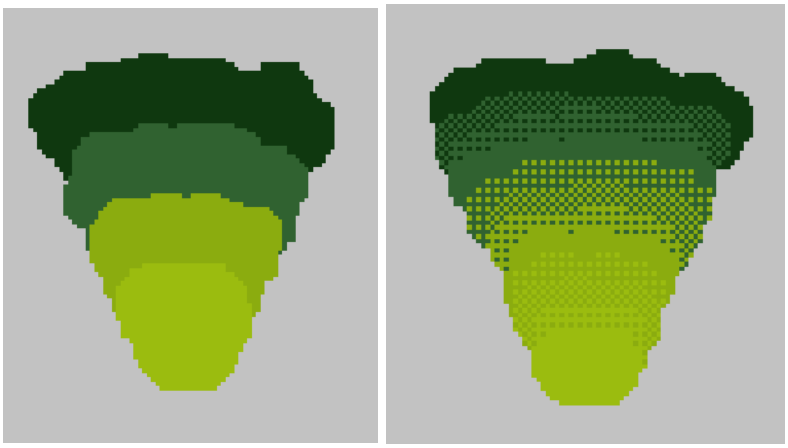


Figura 5.24: *Comparativa entre partícula amb shader de paleta i amb shader de dithering amb textura 2x2*

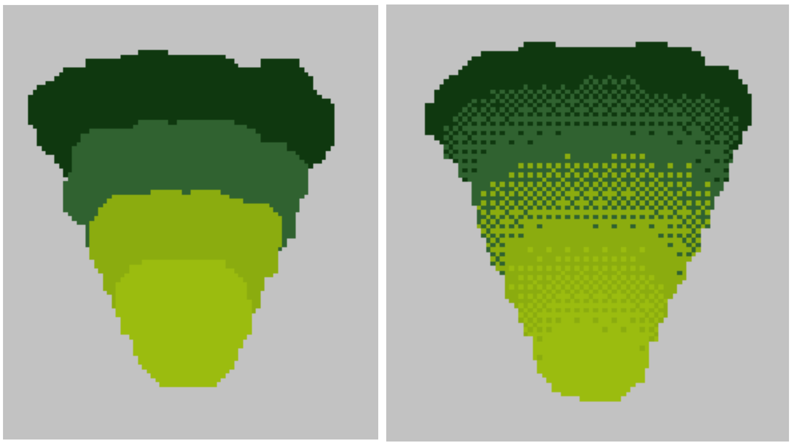


Figura 5.25: *Comparativa entre partícula amb shader de paleta i amb shader de dithering amb textura 4x4*

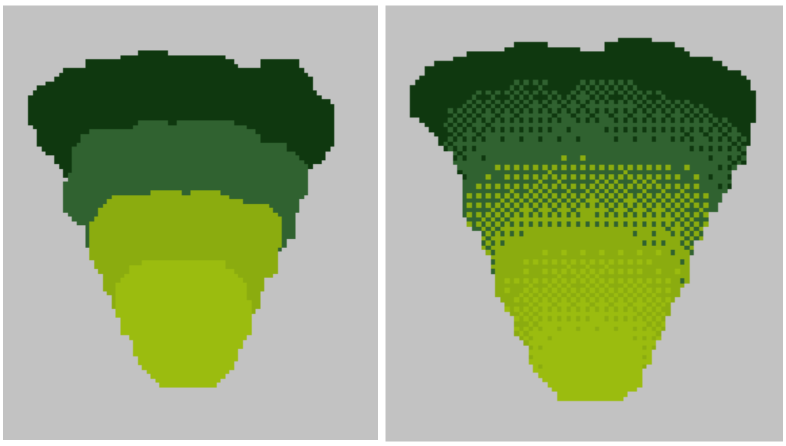


Figura 5.26: *Comparativa entre partícula amb shader de paleta i amb shader de dithering amb textura 8x8*

5.2.6 Antialiasing

La última funcionalitat a desenvolupar és el filtre d'*antialiasing*. L'objectiu d'aquest *shader* és fer una transició suau als colors de l'estil *pixel art* afegint més tonalitats a la paleta. Per implementar-ho s'ha optat per afegir un *shader* de *blur* a la càmera, el qual difuminarà la imatge en pantalla, aconseguit l'efecte desitjat en el renderitzat estil *pixel art*.

```
Properties
{
        [HideInInspector]_MainTex("Texture", 2D) = "white" {}
        _BlurSize("Blur Size", Range(0,0.1)) = 0
        _Samples("Sample amount", Float) = 0
}

...

float _BlurSize;
float _Samples;
```

La propietat a declarar en aquest nou *shader* és un float que indicarà la intensitat de difuminat de pantalla, al qual se l'ha anomenat _BlurSize. També s'ha afegit un valor float que determinarà la quantitat de mostres a tenir en compte en el filtre.

```
fixed4 frag(v2f i) : SV_Target
{
        float invAspect = _ScreenParams.y / _ScreenParams.x;
        float4 col = 0;
        for (float index = 0; index < _Samples; index++)
        {
                float2 uv = i.uv + float2((index / (_Samples - 1) - 0.5) *
                _BlurSize* invAspect, (index / (_Samples - 1) - 0.5) *
                _BlurSize* invAspect);

                col += tex2D(_MainTex, uv);
        }

        col = col / _Samples;

        return col;
}
```

El primer pas a realitzar en aquest *shader* és obtindré el valor invertir de la relació d'aspecte de la pantalla, el qual es farà servir després.

Iterant tantes vegades com mostres hagi determinat l'usuari i aplicant els càlculs que s'observen al codi s'obtindran diferents posicions de pantalla que es sumaran per així obtindré un color mitjà de totes les mostres iterades, aconseguint així l'efecte de difuminat que es buscava.

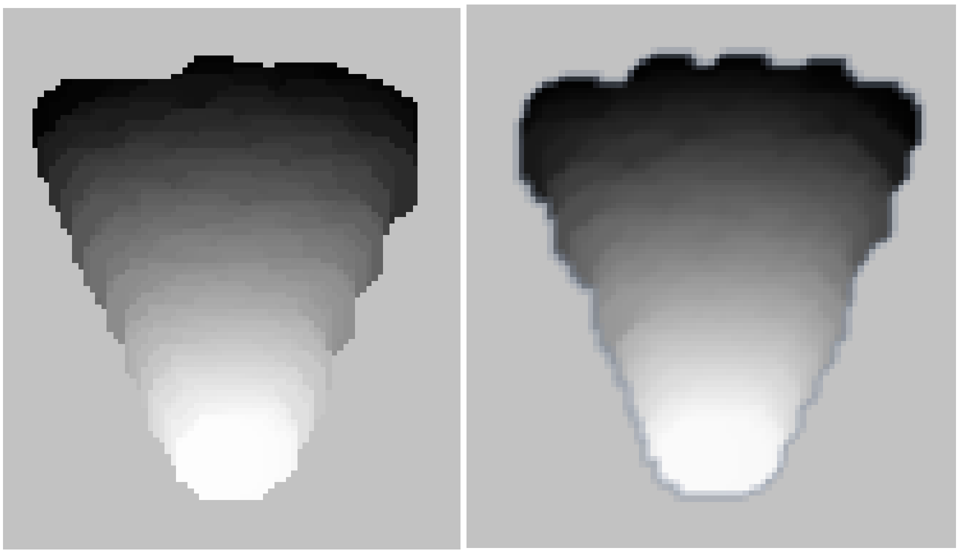


Figura 5.27: Comparativa entre partícula amb render normal i shader antialiasing

5.3 Testing

5.3.1 MVP

La primera sessió de *testing* del projecte va ser realitzada amb una versió que incloïa les següents funcionalitats:

- Edició del sistema de partícules
- Sistema de capes
- Edició del color base de les partícules
- Exportació en format *sprite sheet* .png

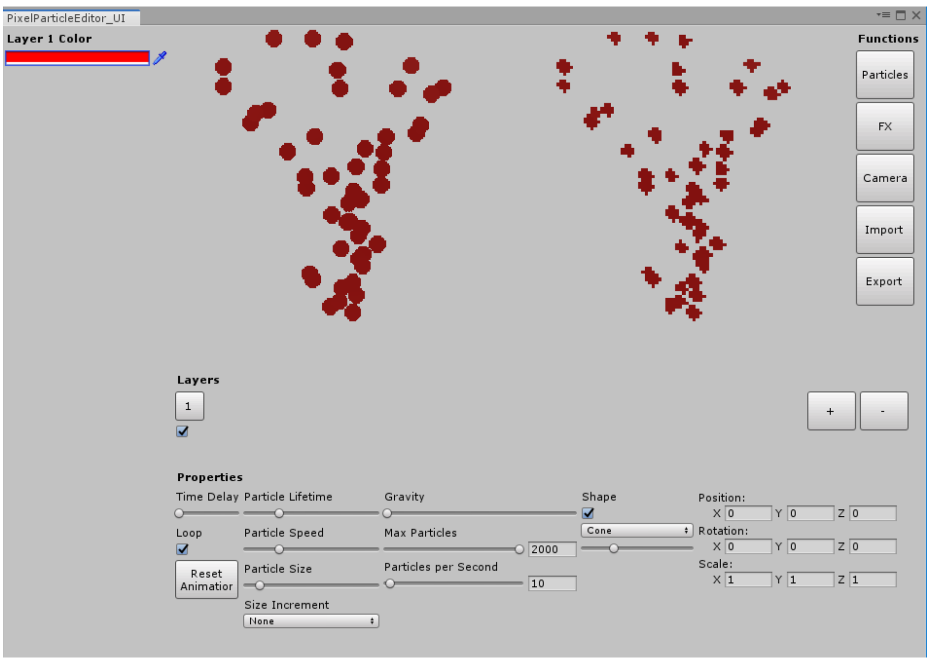


Figura 5.28: Captura de pantalla de la versió MVP de l'asset desenvolupat

L'objectiu d'aquesta sessió va ser confirmar si el desenvolupament de l'*asset* estava complint els objectius proposats o, en cas contrari, replantejar el disseny inicial i modificar els elements de la UI.

El *testing* va constar de cinc proves de 15 minuts realitzades a distància. Els usuaris, en acabar la prova, havien d'omplir el formulari inclòs a l'annex 8.1.1 del document per així donar *feedback* del funcionament i del futur de l'*asset*. Els resultats del formulari van ser els següents:

- ***Quant fàcil has trobat crear una partícula des de zero amb aquest asset? (1 - Molt difícil, 5 - Molt fàcil)***

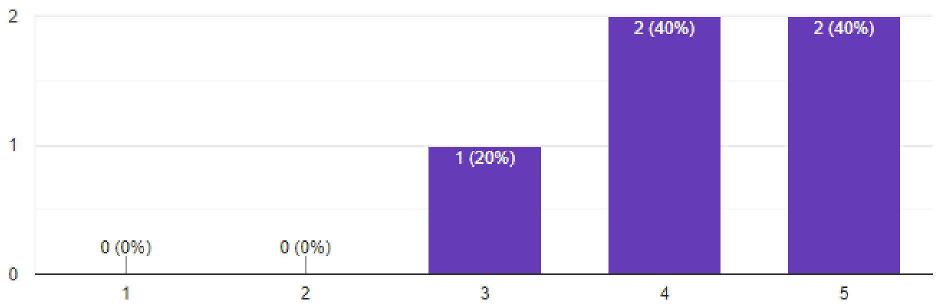


Figura 5.29: *Resultats de la pregunta 1 del testing de la versió MVP de l'asset*

Segons aquestes respostes es va observar que un dels objectius principals del projecte s'estava aconseguint, el qual era el de poder oferir una forma fàcil i entenedora de crear partícules dins de Unity. Per aquesta raó es va continuar treballant amb el disseny inicial, amb l'objectiu de poder obtenir un bon *feedback* en la sessió de *testing* de la versió Beta.

- ***Quanta dificultat has tingut en trobar i entendre les funcionalitats de l'asset? (1 - Molt difícil, 5 - Molt fàcil)***

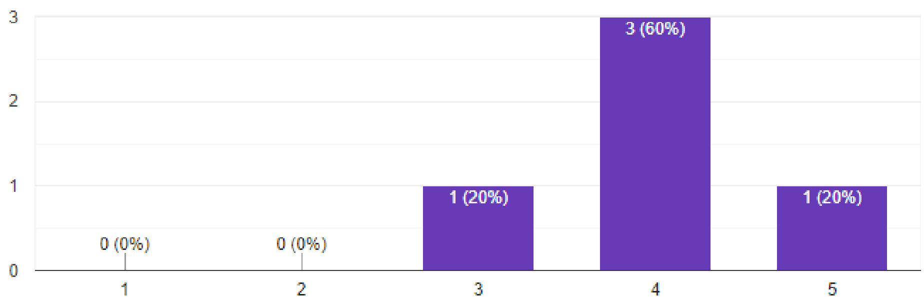


Figura 5.30: *Resultats de la pregunta 2 del testing de la versió MVP de l'asset*

Si bé les respostes a aquesta qüestió van ser molt positives no hi havia una majoria d'usuaris que valoressin al màxim la facilitat d'ús de les funcionalitats. Per solucionar aquest problema es va decidir incloure exemples de partícules en futures versions per mostrar a l'usuari com funciona cada apartat de la interfície.

- ***Quant pagaries en cas d'haver de comprar una versió comercial d'aquest asset?***

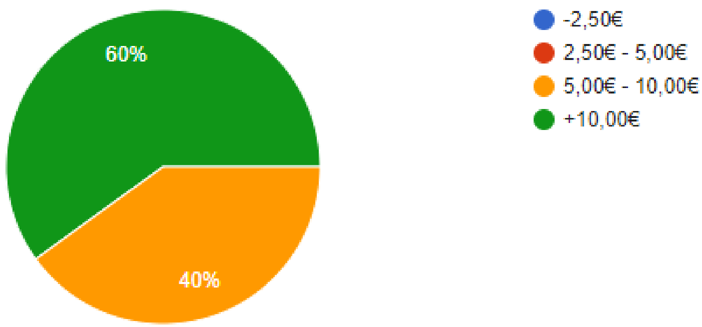


Figura 5.31: *Resultats de la pregunta 3 del testing de la versió MVP de l'asset*

Dels usuaris participants en aquesta sessió tres pagarien més de 10€ per la versió comercial de l'*asset*, mentre que dos pagarien entre 5€ i 10€. Aquestes dades es van tenir en compte a l'hora de decidir el preu de l'*asset* tal i com s'explicarà al punt 5.4.

- ***Quina funcionalitat afegiries en futures versions de l'asset?***

D'entre les funcionalitats demanades, es destaquen les següents:

Noves funcionalitats d'edició de color: Si bé dins de la planificació es van plantejar opcions de paletes personalitzades aquestes només serveixen pels usuaris centrats en crear partícules estil *pixel art*. Per ampliar l'edició de color de l'editor es va prioritzar l'addició de degradats a les partícules i la inclusió del *color blending* a la versió Beta del producte.

Elements d'ajuda: Un dels usuaris va demanar la inclusió d'elements que ajudin a comprendre més ràpidament el funcionament de cada element de l'*asset*. Tot i que dins de la planificació dins del marc del TFG no s'ha pogut incloure cap *sprint* dedicat a elements d'aquest tipus s'intentarà solucionar amb l'addició de les ja esmentades partícules predefinides.

5.3.2 Beta

A la versió final d'aquest projecte es va realitzar una segona sessió de *testing* a la qual van participar tres usuaris. Les sessions van ser presencials i van tenir durades d'entre 30 minuts i 60 minuts, a les quals es van provar totes les funcionalitats i cada usuari va realitzar la seva pròpia partícula. A més, cada usuari va omplir el formulari inclòs a l'annex 8.1.2 per determinar el seu nivell quant a *pixel art* i creació de partícules i efectes especials a Unity.

L'objectiu de la sessió és obtenir *feedback* tant de *bugs* com de la experiència d'usuari del producte. Els resultats d'aquest *testing* són els següents:

Usuari 1:

- **Coneixement de *pixel art:*** 2/5
- **Coneixements en creació de partícules a Unity:** 3/5
- **Duració de la sessió:** 30 min

Aquest usuari entraria dins del *target* mig de l'*asset*, ja que compta tant amb coneixements de Unity i el seu sistema de partícules com també algunes nocions de *pixel art*. Durant la sessió va destacar la rapidesa a l'hora de crear partícules en comparació amb el sistema de partícules original de Unity, a més de la facilitat d'ús de totes les funcionalitats de l'*asset*.

Com a *feedback* de cara a futures versions va plantejar afegir opcions al sistema de capes, com per exemple duplicar-les, moure-les o eliminar-les de manera independent.

També va destacar que la funció que més li va costar entendre va ser la de paletes personalitzables, problema que també es va donar en una altra sessió i es té en compte de cara al futur del projecte.

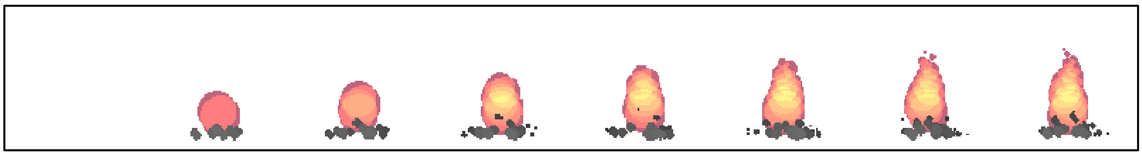


Figura 5.32: *Partícula creada per l'Usuari 1 del testing de la versió Beta*

Usuari 2:

- **Coneixement de *pixel art:*** 1/5
- **Coneixements en creació de partícules a Unity:** 1/5
- **Duració de la sessió:** 60 min

Aquesta va ser la sessió més útil de cara a la l'ús intuïtiu de l'editor per a usuaris novells, ja que l'usuari participant no tenia cap coneixement en els sectors que involucren aquest projecte. Al principi de la sessió li va costar entendre el funcionament de l'*asset*, tot i que va destacar que era problema de no estar familiaritzat amb la nomenclatura del sistema de partícules de Unity, que es la que es fa servir a l'editor.

A mesura que va entendre el funcionament de les partícules va poder crear la seva i exportar-la tant en format *sprite sheet* com Game Object, arribant a afegir-la a l'escena del motor Unity. Aquest resultat va ser molt favorable pel que respecta a l'objectiu d'arribar a un *target* principiant, sobretot pel *feedback* donat respecte a la nomenclatura del sistema de partícules.

També va proposar afegir opcions típiques d'editors editors gràfics com Photoshop, com ara una eina de zoom o mida de la finestra personalitzable. Per acabar cap destacar que no va arribar a entendre el funcionament de la paleta de colors, problema també present amb l'usuari anterior.

Figura 5.33: *Partícula creada per l'Usuari 2 del testing de la versió Beta*

Usuari 3:

- **Coneixement de *pixel art:*** 4/5
- **Coneixements en creació de partícules a Unity:** 3/5
- **Duració de la sessió:** 30 min

L'últim *testing* realitzat va ser amb un usuari experimentat en *pixel art* i el sistema de partícules de Unity, pel qual aquest no va tenir problema en entendre el funcionament de l'editor ràpidament.

El seu *feedback* va ser centrat en els resultats estil *pixel art*, dels quals va destacar que per ser gràfics automatitzats tenien una gran qualitat i que es podrien arribar a fer servir en certs projectes en l'estat actual.

Com a propostes de futur va destacar la possibilitat d'afegir una paleta de colors amb més opcions de personalització, tenint en compte més valors apart de la lluminositat de les imatges. També va esmentar la possibilitat d'afegir opcions d'animació i exportació avançades, opcions que es tenen en compte però a mig-llarg termini, una vegada l'*asset* surti de la fase *beta.*

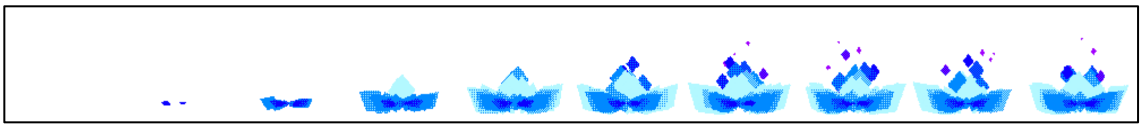


Figura 5.34: *Partícula creada per l'Usuari 3 del testing de la versió Beta*

5.4 Post-Producció

5.4.1 Publicació a l'Asset Store

Una vegada el projecte ha entrat en fase *beta* l'última tasca pendent és publicar l'*asset* a la Store de Unity[16]. El primer pas és iniciar sessió en el lloc web d'editors de Unity, al qual dins la pestanya de "Packages" es troba disponible la opció de crear un nou paquet per la botiga online del motor.

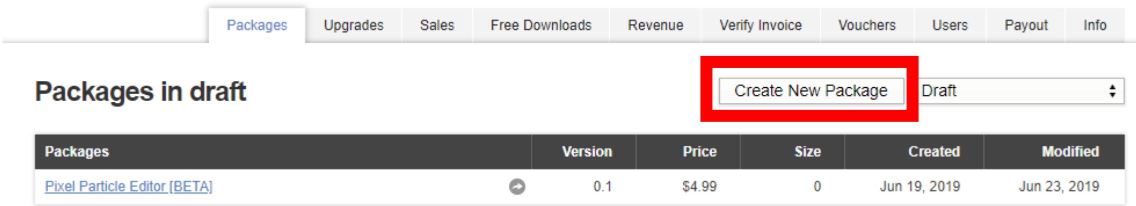


Figura 5.35: *Menú inicial del web d'editors de Unity*

El pas següent a crear el paquet és omplir les caselles de detalls que es troben disponibles dins del borrador. Les dades a omplir són:

- **Versió:** Valor numèric que marcarà la versió a la que es troba l'*asset.*
- **Canvis de Versió:** Canvis aplicats al paquet segons les versions que han hagut d'aquest.
- **Categoria:** Categoria de la botiga online on estarà l'*asset.* En el cas d'aquest projecte estarà en la categoria "Extensions d'Editor/Efectes".
- **Preu:** Preu al que es vendrà l'*asset*. El preu escollit és 4.99€ de manera temporal mentre es trobi en fase *beta.* Quan en un futur surti d'aquesta fase el preu base es pujarà a 14,99€.

Quan s'hagin omplert totes aquestes dades és el torn d'afegir la *metadata* del paquet, és a dir, les dades que descriuen el contingut de l'*asset,* els quals són el nom, la descripció i les paraules clau del projecte. A més, hi ha la opció d'afegir un logo per identificar l'*asset* a la Store de Unity i captures de pantalla per mostrar el funcionament d'aquest.

Per acabar la publicació s'ha de pujar el paquet de l'*asset* des del mateix motor Unity, seguint les instruccions indicades després dels apartats de dades que s'han explicat anteriorment. Una vegada enviat, el paquet serà revisat per l'equip de Unity i en un temps de dues setmanes envien la confirmació o declinació de la sol·licitud. En cas de que sigui aprovat el paquet es trobarà disponible a la Asset Store.

[16] Per poder publicar a l'Asset Store de Unity és necessari crear un compte de Publisher. Tant per crear el compte com per accedir als menús explicats en aquest punt s'ha d'accedir al següent enllaç: https://publisher.assetstore.unity3d.com/

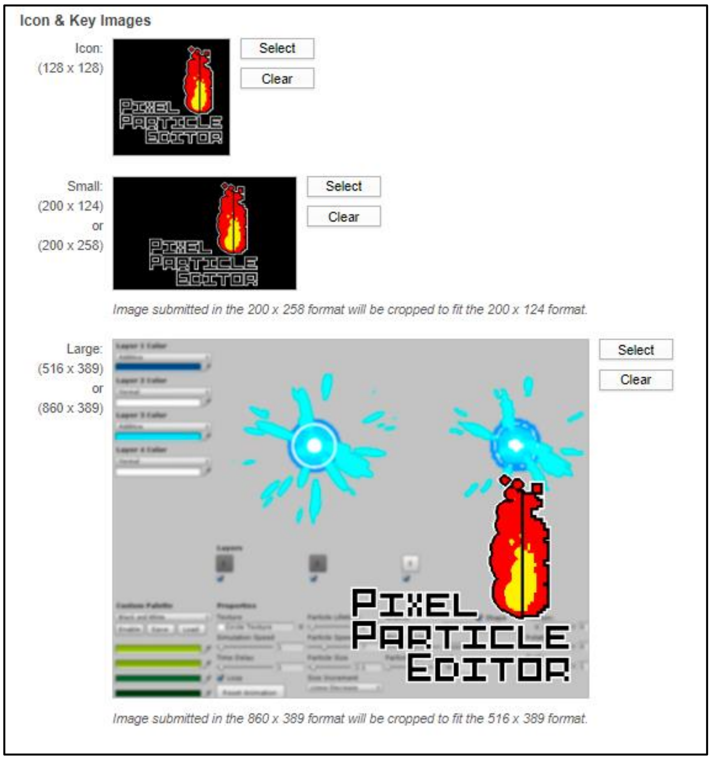


__Figura 5.36:__ Menú de logos i imatges clau al web d'editor de Unity

__Figura 5.37:__ Previsualització de la finestra de compra a l'Asset Store

6. Conclusions i treball futur

6.1 Conclusions del projecte

Els dos grans objectius plantejats a l'inici d'aquest projecte van ser, d'una banda, oferir una experiència intuïtiva que pogués satisfer tant usuaris novells com experimentats, i, per l'altra banda, aconseguir resultats de qualitat en estil *pixel art.* Després del treball realitzat i el *feedback* rebut es podria dir que aquest dos objectius han sigut aconseguits amb èxit, donant com a resultat un producte que es podria utilitzar en projectes petits i agilitzar molt la creació de prototipus i *game jams*. De totes maneres s'ha de tenir en compte que encara queda un gran marge per acabar de polir i perfeccionar els resultats.

Quant a la planificació del treball s'han seguit les dades plantejades, podent realitzar la gran majoria de funcionalitats pensades. Només s'ha descartat la funcionalitat d'exportació en .gif degut al temps limitat d'aquest projecte i la quantitat de tasques pendents per fer al llarg del transcurs del treball. Ha sigut de gran ajuda el canvi d'eina de seguiment, passant de Trello a HacknPlan. Gràcies a aquest canvi es van poder quantificar les hores dedicades a cada tasca individual, ajudant a la planificació d'aquestes setmana rere setmana.

Pel que respecta al futur d'aquest *asset*, es continuarà treballant en ampliar i millorar les seves funcionalitats. Es realitzaran ajustaments en la usabilitat de l'editor, afegint tant millores proposades pels usuaris participants del segon *testing* com d'altres que estiguin presents en la gran majoria d'editors, com per exemple la opció de desfer l'últim canvi realitzat. Per acabar es farà una revisió del aspecte gràfic de la interfície, amb la intenció d'incloure una *UI* estil *pixel art* com la del programa Aseprite.

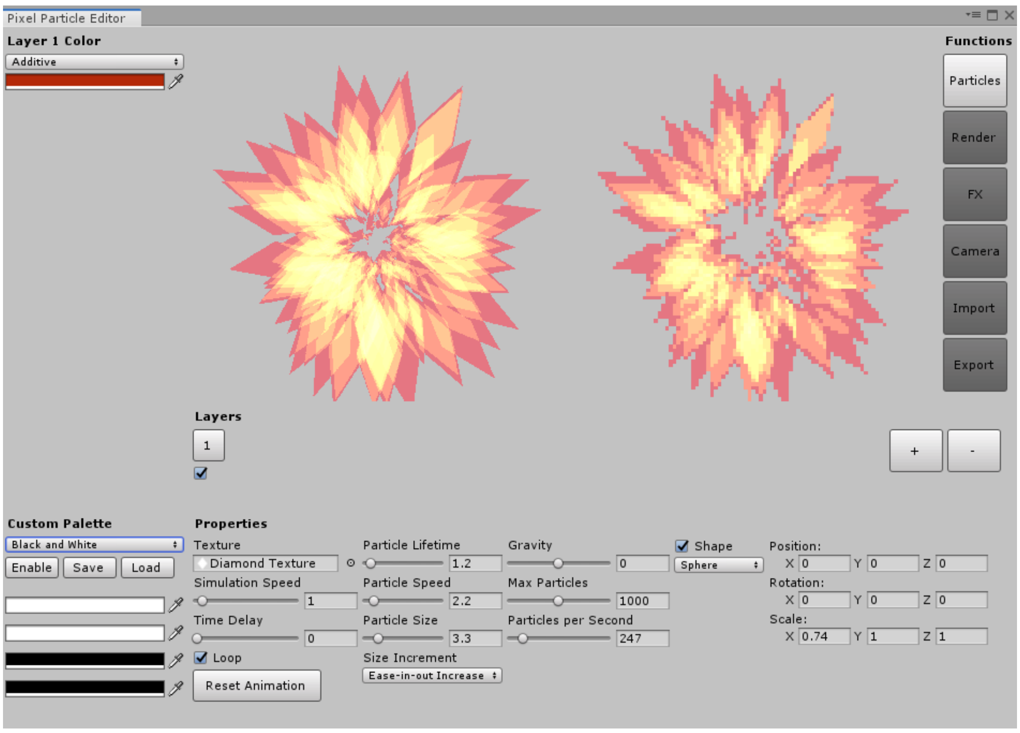


Figura 6.1: *Captura de pantalla de la versió beta de l'asset desenvolupat*

6.2 Conclusions personals

Aquest projecte va néixer com un repte personal que em vaig proposar per posar a prova els meus coneixements de *pixel art* i programació, i el primer que m'agradaria destacar és, sense dubte, com aquest treball m'ha ajudat a créixer en tant en aquests camps com en d'altres.

He pogut aplicar molts dels coneixements apresos al grau, ja que en les diferents etapes del projecte m'he trobat en situacions on es requerien coneixements de cadascun dels àmbits. Per començar s'ha requerit coneixements d'art i programació pel desenvolupament de l'*asset.* A l'hora de dissenyar la interfície s'han aplicat coneixements d'usabilitat i experiència d'usuari, i per acabar s'han tingut en compte alguns elements de màrqueting a l'hora de posar els preus i la informació del producte a l'Asset Store. Gràcies a aquest punt de partida espero poder seguir aplicant tot l'après en futurs treballs i projectes.

L'obstacle que més ha costat superar és, sense dubte, la programació de *shaders*, aspecte que em va obligar a fer una cerca exhaustiva tant de la informació disponible a la documentació i exemples pràctics aplicats en projectes similars. Després del treball realitzat puc assegurar que el desenvolupament de *shaders* ha despertat el meu interès i espero poder seguir realitzant projecte i créixer en aquest àmbit.

Espero poder continuar desenvolupant videojocs al llarg de molt de temps, en especial si compten amb estètica *pixel art* pel significat que te per a mi aquest estil visual, i estic segur que la creació d'aquest *asset* serà el punt de partida de tot el que estigui per arribar.

7. Bibliografia

- Sonic Retro – RotSprite. *info.sonicretro.org/RotSprite*. Consultat. 11 Mar, 2019
- Pixel FX Designer. *codemanu.itch.io/particle-fx-designer*. Consultat. 11 Mar, 2019
- The Factory Times – The History of Pixel Art. *www.thefactorytimes.com/factory-times/2018/9/27/the-history-of-pixel-art*. Consultat. 11 Mar, 2019
- 2DForever – Fix Your Pixel Art. *2dforever.com/fix-your-pixelart*. Consultat. 11 Mar, 2019
- Medium – Pedro Medeiros. *medium.com/@saintjust*. Consultat. 12 Mar, 2019
- Les Vendredis Webdesign – Démystifions le Pixel Art : Interview d'Oliver Huard *wdfriday.com/wdfriday.com/blog/2012/10/demystifions-le-pixel-art-interview-dolivier-huard*. Consultat. 12 Mar, 2019
- The Verge - Pixel art games aren't retro, they're the future. *www.theverge.com/2014/7/3/5865849/pixel-art-is-here-to-stay*. Consultat. 12 Mar, 2019
- IGN – 10 Best NES Games of All Time. *www.ign.com/articles/2018/11/25/10-best-nes-games-of-all-time*. Consultat. 12 Mar, 2019
- The Gemsbok - FTL: Faster Than Light, and Pixel Art as an Art Movement. *thegemsbok.com/art-reviews-and-articles/video-game-reviews-mid-week-mission-ftl-faster-than-light-subset-games*. Consultat. 12 Mar, 2019
- Hackernoon - Making your Pixel Art Game look Pixel Perfect in Unity3D. *hackernoon.com/making-your-pixel-art-game-look-pixel-perfect-in-unity3d-3534963cad1d*. Consultat. 12 Mar, 2019
- Slynyrd – Pixelblog – 1 – Color Paletters. *www.slynyrd.com/blog/2018/1/10/pixelblog-1-color-palettes*. Consultat. 12 Mar, 2019
- Tiny Warrior Games - Game Development: Pixel Art Hue Shifting. *tinywarriorgames.com/2018/12/21/game-development-pixel-art-hue-shifting*. Consultat. 12 Mar, 2019
- Pixel Art Intro. cfkaligula.blogspot.com/2016/09/pixel-art-intro_5.html. Consultat. 12 Mar, 2019
- Medium – Constructing a Pixel Art Shader (Stephen Schroeder) *medium.com/@thedeivore/constructing-a-pixel-art-shader-3753762f6b90*. Consultat. 12 Mar, 2019

- Lospec – Pixel Art Software List. lospec.com/pixel-art-software-list. Consultat. 13 Mar, 2019

- Concept Art Empire - Best Pixel Art Software For All OS'. conceptartempire.com/pixel-art-software. Consultat. 13 Mar, 2019

- Aseprite. www.aseprite.org. Consultat. 13 Mar, 2019

- PyxelEdit. pyxeledit.com. Consultat. 13 Mar, 2019

- Unity. unity.com. Consultat. 13 Mar, 2019

- James Heazlewood's Programming Blog - Unity Sparks System. seagullcity.wordpress.com/2013/10/13/unity-sparks-system. Consultat. 13 Mar, 2019

- HacknPlan. hacknplan.com. Consultat. 14 Mar, 2019

- GitHub. github.com. Consultat. 14 Mar, 2019

- Google Forms. www.google.es/intl/es/forms/about. Consultat. 14 Mar, 2019

- Indeed - Sueldos en Programador/a junior en España. www.indeed.es/salaries/Programador/a-junior-Salaries?period=monthly. Consultat. 14 Mar, 2019

- Unity – Unity User Manual. docs.unity3d.com/Manual/index.html. Consultat. 15 Abr, 2019

- Prototypr – In defense of skeuomorphism. blog.prototypr.io/in-defense-of-skeuomorphism-2895308218ee. Consultat. 26 Abr, 2019

- Kirill Nadezhdin – Game Boy Processing shader for Unity. medium.com/@cyrilltoboe/code-first-game-boy-post-processing-shader-for-unity-ef140252fd7d. Consultat. 13 Maig, 2019

- NVidia Developer Zone - Cg Standard Library Documentation. developer.download.nvidia.com/cg/index_stdlib.html. Consultat. 14 Maig, 2019

- Ronja Böhringer – Dithering. www.ronja-tutorials.com/2019/05/11/dithering.html. Consultat. 3 Juny, 2019

- Ronja Böhringer – Blur Postprocessing Effect. www.ronja-tutorials.com/2018/08/27/postprocessing-blur.html. Consultat. 6 Juny, 2019

- Unity – Asset Store Publisher. publisher.assetstore.unity3d.com. Consultat. 19 Juny, 2019

8. Annexos

8.1 Enquestes de Validació

8.1.1 Enquesta MVP

Pixel Particle Editor MVP

* Required

Did you find it easy to create a new particle from scratch with this asset? *

	1	2	3	4	5	
Totally Disagree	○	○	○	○	○	Totally Agree

How difficult was to find the functions of the asset? *

	1	2	3	4	5	
Really Hard	○	○	○	○	○	Really Easy

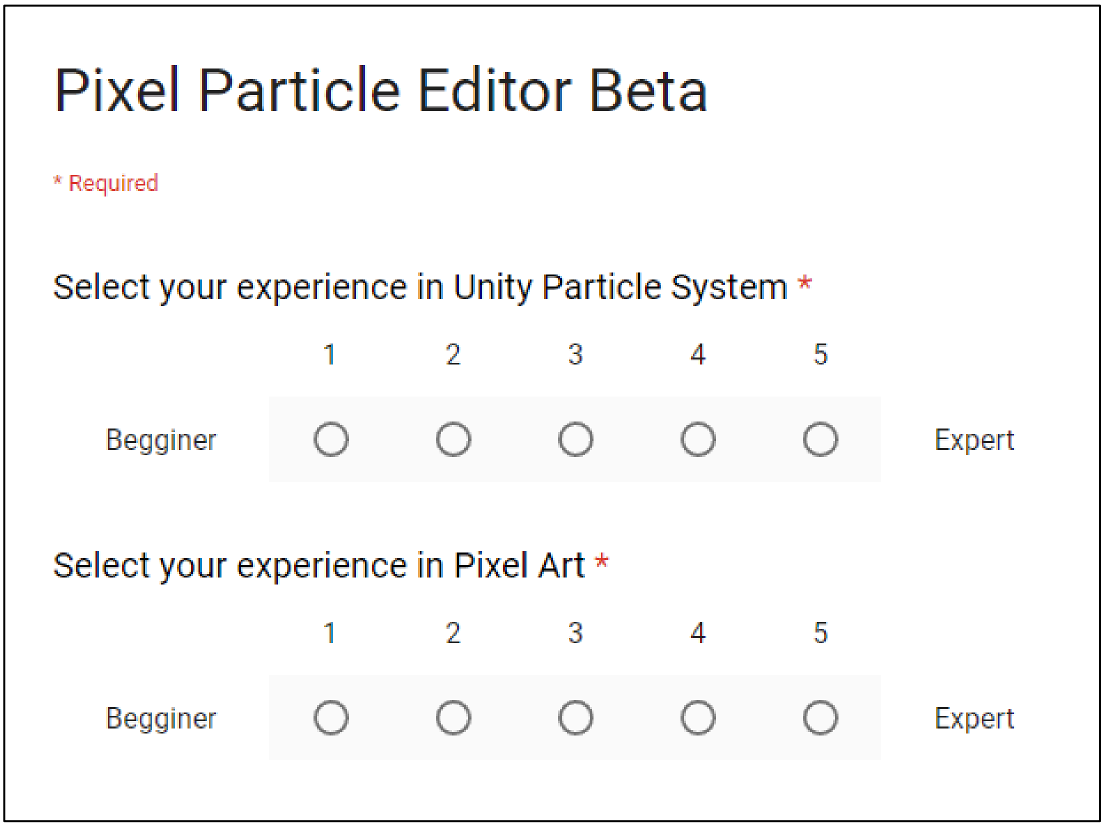
How much would you pay for this asset if it was available in the Asset Store?

- ○ +10,00€
- ○ 5,00€ - 10,00€
- ○ 2,50€ - 5,00€
- ○ -2,50€

What feature would you add in the next release of this asset?

Your answer

8.1.2 Enquesta Beta



Pixel Particle Editor Beta

* Required

Select your experience in Unity Particle System *

	1	2	3	4	5	
Begginer	○	○	○	○	○	Expert

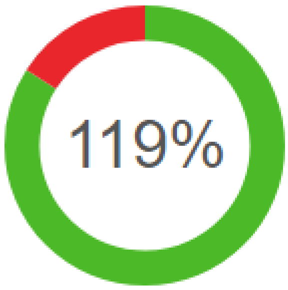
Select your experience in Pixel Art *

	1	2	3	4	5	
Begginer	○	○	○	○	○	Expert

8.2 Quantificació de temps de treball

*No inclou el temps dedicat al disseny de les funcionalitats i l’estudi de les tecnologies necessàries pel seu desenvolupament.

8.2.1 Sprint MVP

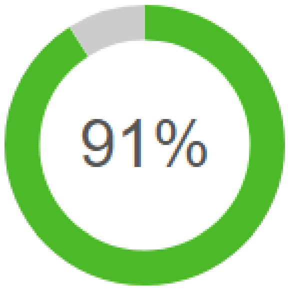


HOURS

Estimated	Logged	Remaining
10h 15m	12h 10m	0

	Estimated	Logged	Unused
USER STORY	0	0	0
SHADER	2h	1h 30m	30m
UI	4h	6h 15m	- 2h 15m
DESIGN	0	0	0
MARKETING	0	0	0
IDEAS	0	0	0
BUG	0	0	0
IMPORT/EXPORT FEATURE	3h 45m	4h 15m	- 30m
FX FEATURE	30m	10m	20m
	10h 15m	12h 10m	- 1h 55m

8.2.2 Sprint Palette

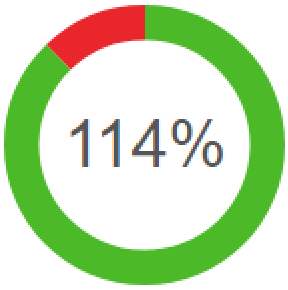


HOURS

Estimated	Logged	Remaining
11h 15m	**10h 15m**	**0**

	Estimated	Logged	Unused
USER STORY	0	0	0
SHADER	6h 45m	5h 30m	1h 15m
UI	2h	1h 15m	45m
DESIGN	0	0	0
MARKETING	0	0	0
IDEAS	0	0	0
BUG	1h	1h 30m	- 30m
IMPORT/EXPORT FEATURE	1h 30m	2h	- 30m
FX FEATURE	0	0	0
	11h 15m	10h 15m	1h

8.2.3 Sprint Dithering & Antialiasing



HOURS

Estimated	Logged	Remaining
5h 30m	6h 15m	0

	Estimated	Logged	Unused
USER STORY	0	0	0
SHADER	3h	4h	- 1h
UI	30m	45m	- 15m
DESIGN	0	0	0
MARKETING	0	0	0
IDEAS	0	0	0
BUG	2h	1h 30m	30m
IMPORT/EXPORT FEATURE	0	0	0
FX FEATURE	0	0	0
	5h 30m	6h 15m	- 45m