Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Disseny i Desenvolupament de Videojocs

Títol: Creació d'un joc que barregi mecàniques de Survival, Tower Defense i Modelatge procedural.

Document: Memòria

Alumne: Josep Saavedra Martínez

Tutor: Gustavo Patow
Departament: Informàtica, Matemàtica Aplicada i Estadística
Àrea: Llenguatges i sistemes informàtics

Convocatòria (mes/any): Juny 2023

ÍNDEX

1. Introducció i objectius....6
1.1.Introducció....6
1.2.Motivacions.... 6
1.3.Propòsit i objectius....6
1.4.Distribució de tasques....7
2.Estudi de viabilitat.... 8
2.1. Recursos....8
2.1.1. Software.... 8
2.1.1.1. Motor.... 8
2.1.1.2. Llenguatge....9
2.1.1.3. Extres.... 9
2.1.1.4. Altres opcions....10
2.1.2. Recursos Humans.... 11
2.1.2. Pressupostos....11
2.2. Estudi de Mercat.... 12
2.2.1. Estat de l'art.... 12
2.2.1.1. Noita....12
2.2.1.2. Muck....13
2.2.1.3. Minecraft....13
2.2.1.4 Rogue Towers.... 14
2.2.1.5 Factorio.... 14
2.2.2 Taula Comparativa.... 15
2.2.3 Resultats.... 16
2.3. Públic objectiu....16
3. Planificació.... 17
3.1. Pla de Treball.... 17
3.2. Metodologia.... 18
3.3. Resultats esperats.... 19
● 4. Marc del treball....20
4.1. Referències....20
4.1.1 Bloons TD 6....20
4.1.2 Rogue Towers.... 21
4.1.3 Muck....21
4.1.4 Risk of Rain 2....22
4.2 Motor de videojoc.... 22
4.2.1 Unity 3D....22

4.3. Eina de disseny 3D........ 23
4.3.1 Blender........ 23
4.4. Equip de treball........ 24
4.5. Conceptes Previs........ 24
4.5.1. Vocabulari anglès utilitzat........ 24
4.5.2. Editor........ 25
4.5.3. Scripts........ 26
4.5.4. Col·lisions i triggers........ 27
4.5.5. Raycast i SphereCast........ 27
4.5.6. Sistema d’esdeveniments........ 28
4.5.7. Mesh i Materials........ 28
4.5.7. Nav Mesh........ 29
4.5.8. Prefabs........ 30
4.5.9. Canvas i interficies........ 30
4.5.10. Scriptable object........ 30
5. Disseny del videojoc........ 32
5.1. Objectius de disseny........ 32
5.2. Disseny del mapa........ 32
5.3. Mecàniques........ 33
5.3.1. Accions del personatge........ 33
5.3.1.1 Energia........ 33
5.3.1.2. Moviment........ 34
5.3.1.2. Atac i recol·lecció........ 34
5.3.1.3. Construcció........ 34
5.4. Gestió de recursos........ 34
5.5. Cicle nit i dia........ 35
5.6.Menú........ 36
5.6.1. Menú principal........ 37
5.6.2 Menú de pausa........ 37
5.6. Reptes i objectius........ 38
5.7. Estètica........ 38
5.7.1 Models 3D........ 39
5.7.2 Interfície........ 40
5.8. Elements a desenvolupar........ 42
5.8.1 Menú........ 42
5.8.1 HUD........ 42
5.8.2 Models 3D........ 42
5.8.3.Generació procedural........ 43
5.8.4.Cicle nit i dia........ 43
5.8.5. Mecàniques........ 43
5.9. Objectius descartats o per a un futur........ 43
6. Implementació........ 45
6.1 Objectes destructibles........ 45
6.2 Cicle dia i nit........ 49

6.2.1 Gradients i percentatges.. 49
6.2.2 Implementació.. 49
6.3 Editor.. 52
6.4 Enemics.. 53
6.4.1 Directors i spawn cards.. 53
6.4.2 Nav Mesh.. 54
6.4.3 Implementació enemics..56
6.4.4 Gestió de rondes.. 62
6.4.5 Implementació director.. 63
6.4.6 Mostrar la vida actual.. 66
6.5 Sistema d'esdeveniments.. 68
6.5.1. Observer..68
6.5.2. Implementació.. 69
6.6 Handler.. 71
6.6.1 Control del cursor.. 71
6.6.2 Estat de la partida.. 72
6.7 HUD.. 74
6.8 Objectes interactius.. 76
6.9.1. Menú principal.. 78
6.9.2. Dins de la partida.. 80
6.10.Sistema de millora.. 83
6.10.1 Arrossegar objectes.. 83
6.10.2 Chip.. 86
6.10.2.1. Chips de millora de chips.. 89
6.10.2.2. Chips de millora de estructures..91
6.10.2.3. Visualització dels chips.. 92
6.10.3. Sockets..93
6.10.3.1 Estació..93
6.10.3.2 Socket.. 96
6.10.4. Estadístiques..99
6.10.4.1 Implementació.. 99
6.10.4.2 Interficie..101
6.11.Jugador.. 104
6.11.1. Estadistiques.. 104
6.11.2. Moviment.. 107
6.11.3. Eines.. 110
6.11.3.1. Pic.. 112
6.11.3.2. Constructor.. 115
6.12. Gestió de recursos.. 116
6.12.1 Recursos.. 116
6.12.2 Inventari.. 119
6.12.3 Receptes.. 121
6.13. Estructures defensives..126
6.13.1. Escollir objectiu.. 127

6.13.2. Atacar.......... 128
6.13.3. Projectil.......... 130
6.13.3.1. Comportament.......... 130
6.13.3.2. Efectes.......... 132
6.14. Generació procedural.......... 133
6.14.1 Noise.......... 133
6.14.2 Mesh.......... 137
6.14.3 Textura.......... 139
6.14.4. Visualització.......... 140
6.14.5. Color per regions.......... 140
6.14.6. Implementació.......... 141
6.14.7. Generació de recursos.......... 142
6.15. Extres.......... 143
7. Resultat.......... 145
7.1. Captures de Pantalla.......... 146
8. Conclusions.......... 149
9. Treball Futur.......... 150
10. Bibliografia.......... 151
11. Annexos.......... 152
12. Manual d’usuari i instal·lació.......... 153
12.1 Manual d'instal·lació.......... 153
12.2 Manual d’usuari.......... 153
12.3 Objectiu.......... 153

1. Introducció i objectius

1.1.Introducció

En l'àmbit dels videojocs, els 2020 van aportar un creixement exponencial dels jocs on, en comptes d'una història o narrativa complexe, els dissenyadors elegeixen crear mecàniques on el jugador pot elegir com enfrontar-se als conflictes, partides curtes d'entre 30 a 2 hores on la aleatorietat i dificultat formen la base de l'experiència. Jocs coneguts com *RogueLike* o *RogueLite, o*n en un Roguelike, o joc similar a Rogue*, la mort és permanent, en un Rogue-lite el jugador pot obtenir objectes per a partides futures. Aquest estil de videojoc té moltes varietats: Acció, cartes, shooters, etc, però pocs utilitzen la construcció i millora de defenses com a mecànica principal, i dels pocs que hi han, s'utilitza molt poc la visió en primera persona.

1.2.Motivacions

Aquest projecte té les següents motivacions principals:

- Gran interés en la generació procedural
- Coneixement sobre la creació de jocs que utilitzen aleatorietat.
- Aprendre més sobre com construir mecàniques interessants i complexes.
- Experimentació del motor de Unity de manera més profunda
- La creació d'un prototip jugable que pugui convertirse en un videojoc.
- Millorar els coneixements de C#

1.3.Propòsit i objectius

El propòsit del projecte és la creació d'un prototip jugable d'un joc de supervivencia procedural, on el jugador haurà de sobreviure rondes d'enemics aleatoris mentre construeix defenses i les millora amb un sistema de modificació basat en nodes. El joc consisteix en explorar un mapa creat proceduralment i obtenir recursos amb els quals construir defenses per protegir-se de les onades d'enemics que apareixen, aquests també de manera aleatoria.

El prototip utilitzarà Unity3D i les parts principals seran:

- Generar mapa i recursos procedurals
- Generar ones d'enemics aleatories

- Crear un sistema d'inventari
- Crear un sistema de construcció i millores
- Expandir el coneixement del llenguatge C# i patrons de disseny de programació.

1.4.Distribució de tasques

El projecte és principalment un prototip tècnic amb l'objectiu d'obtenir les bases del que seria un joc complet. Per tant hi ha hagut molt d'enfoc a les mecàniques, especialment la generació procedural i el sistema de millores. Cal tenir en compte que el projecte ha estat realitzat per una persona.

Estètica	10%
Narrativa	0%
Mecàniques	60%
Tecnologia	30%

Figura 1.1. Distribució de tasques

2.Estudi de viabilitat

En aquest apartat mirarem els recursos necessaris, tant de software com humà, per a dur a terme el projecte. A més, mirarem la seva posició en el mercat per comprovar si hi ha un espai pel producte final.

2.1. Recursos

La creació d’un videojoc requereix una gran quantitat de recursos. A continuació mostrarem els recursos que s’utilitzaran el projecte i aquells que s’han considerat, pero no s’utilitzaran.

2.1.1. Software

A la hora d’elegir quin software utilitzar, s’ha decidit elegir els recursos que combinin un preu assequible, o gratuït, amb experiencia previa.

2.1.1.1. Motor

Figura 2.1 Logotip de Unity

El motor escollit pel projecte és Unity3D. Actualment és de les millors opcions per desenvolupar videojocs gratuïtament i proporciona les eines requerides per dur a terme l’objectiu final. A més d’una gran quantitat d’assets gratuïts i de lliure ús en la seva botiga.

2.1.1.2. Llenguatge

Figura 2.2. Logotip de C#

El llenguatge de programació utilitzat és el que utilitza el motor Unity3D, el C#. Especificament la versió 2021.2.3f1

2.1.1.3. Extres

Figura 2.3. Logotip de Blender

Eina gratuita per crear models 3D de manera ràpida i compatible amb Unity3D.

Figura 2.4. Logotip d’Ableton

Estació de treball d'àudio digital (EAD), eina que s’utilitzarà per afegir música al projecte.

Figura 2.5. Logotip de GitHub

S’ha utilitzat principalment per controlar versions i com a backup i emmagatzematge.

2.1.1.4. Altres opcions

Figura 2.6. Logotip de Unreal Engine

Unreal Engine és un motor de videojocs especialitzat en videojocs AAA, amb eines per equips grans. No s’ha elegit per la mida del projecte i de la falta d'experiència en aquest motor.

Figura 2.7. Logotip de Godot

Godot és un motor de videojocs que s'enfoca més al 2D, malgrat que es poden crear projectes en 3D. No s'ha elegit com a motor principal ja que és un motor relativament nou i no té suficients eines per a ser viable i còmode d'utilitzar.

2.1.2. Recursos Humans

La producció d'un videojoc requereix un equip format per persones de diversos àmbits: programació, marketing, mùsica, disseny 3D, artistes, etc. Malgrat això, el projecte serà realitzat per una persona.

2.1.2. Pressupostos

Unity no cobren si el benefici de l'any fiscal anterior no supera un valor de 100.000 USD anuals. Al ser un projecte sense ànim de lucre, podem utilitzar Unity gratuïtament.
Únicament hem de tenir en compte el preu de les eines de creació de mùsica, Ableton i un Teclat, i el hardware. En aquest cas, com és l'ordinador i teclat personal de l'autor, contem que aquest és 0 actualment en aquest projecte. Una idea general del cost hipotètic seria el següent:

Recurs	Cost(€)
Ordinador personal	2000
Teclat de mùsica	222
Altres	0

Figura 2.6. Quadre de preu de recursos

2.2. Estudi de Mercat

Ara que tenim la idea i els recursos, hem d'estudiar el mercat i comprovar si hi ha espai pel nostre producte. Al ser un joc dividit en dues parts, supervivencia amb obtenció de recursos i tower defense de mecàniques complexes hem de mirar els jocs que tinguin una de les dues parts, o que combinin les dues.

2.2.1. Estat de l'art

Com hem dit anteriorment, hem de buscar jocs que combinin la supervivència i gestió de recursos amb mecàniques complexes.

2.2.1.1. Noita

Figura 2.2.6. Logotip de Noita

Noita es un videojoc 2D on el jugador controlarà a un alquimista que ha d'explorar unes coves en busca de secrets. Noita obliga al jugador a adaptarse en terrenys hostils amb els recursos i magies que trobi pel camí. Les parts més innovadores del videojoc són un revolucionari sistema de simulació de pixels, aquesta part no afecta al nostre projecte, I un sistema extremadament complex i profund de creació de vares màgiques. (Veure figura 2.2.6)

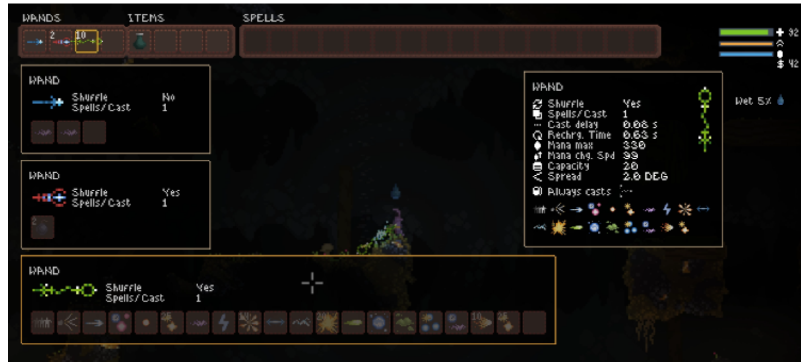

Figura 2.2.6. Menú de creació de Noita

2.2.1.2. Muck

Figura 2.2.6. Logotip de muck

En Muck, el jugador haurà d’explorar una illa creada proceduralment buscant i utilitzant recursos per crear eines i escapar. Podrà fer ús d’espases, arcs, pistoles i podra obrir cofres amb millores pel personatge.
Aquest joc és la principal inspiració del projecte.

2.2.1.3. Minecraft

Figura 2.2.7. Logotip de Minecraft

El segon videojoc mes comprat de la història amb 238 milions de compres. Minecraft és un videojoc enfocat a la construcció i creació d'objectes. Els mapes són creats proceduralment.

2.2.1.4 Rogue Towers

Figura 2.2.8. Logotip de Rogue Tower

Rogue towers és un Rogue Lite on el jugador haurà de construir torres i millorarles a partir de cartes obtingudes aleatòriament. El joc combina mecàniques de roguelikes amb tower defense per una experiencia ràpida i simple.
El joc s'enfoca en debilitats elementals i sinergies entre cartes. Amb un enfoc molt elevat en la gestió de recursos.

2.2.1.5 Factorio

Figura 2.2.9. Logotip de Factorio

Factorio és un videojoc del gènere "Construcció de fàbriques". Enfocat principalment en la gestió de recursos, automatització i enginyeria. El jugador haura de construir una fábrica i obtenir suficients recursos per construir una nau espacial i escapar del planeta, mentre animals extraterrestres intenten destruir al jugador i les seves construccions.
La dificultat del joc recau en la automatització i complexitat de les creacions. Encara que té parts de tower defense, aquesta part és simple i no l'objectiu principal.

2.2.2 Taula Comparativa

Agafem les parts més característiques i comparem-les.(veure taula 2.2.10; 2.2.11)

Videojoc	Gestió de recursos	Mapa procedural	Defensar amb torres	Mécaniques complexes	Gràfics	Preu(€)
Noita	No	Si	No	Si	Frontal 2D	19
Muck	Si	Si	No	No	FPS 3D	Gratuït
Minecraft	Si	Si	Possible	Si	FPS 3D	39.99
Rogue Towers	No	Si	Si	No	Isometric 3D	12.49
Factorio	Si	Si	Si	Si	Top down 2D	32

Figura 2.2.10. Taula de característiques del mercat

Videojoc	Gestió de recursos	Mapa procedural	Defensar amb torres	Mécaniques complexes	Gràfics	Preu(€)
Prototip	Si	Si	Si	Si	fps 3D	Gratuït

Figura 2.2.11. Taula de característiques del projecte

2.2.3 Resultats

Com podem veure a la taula, tots els videojocs tenen parts que comparteixen amb el nostre projecte. Els 2 videojocs més similars serien Factorio i Minecraft, amb Muck com menció d'honor ja que també és en 3a persona. Malgrat això, el projecte és diferència en les següents característiques:

- Minecraft està enfocat en la creativitat i construcció, mentre que el projecte es basa en la supervivència i eficiencia de les defenses que posa el jugador.
- El nostre joc conta amb un sistema de millores de torre original, on el jugador podrà controlar el preu i poder d'aquestes. Això el fa únic respecte a Rogue Tower i altres Rogue lites que depenen d'un gran factor d'aleatorietat per a les millores.

La conclusió sobre aquestes dades és que el nostre joc és suficientment distintiu per tenir un espai en el mercat, però també és suficientment similar a altres jocs per guanyar l'interès dels jugadors.

2.3. Públic objectiu

El nostre videojoc serà una experiencia ràpida amb partides entre 30 minuts i 2 hores. Per tant, el perfil que busquem seria d'una persona, adolescent o adulta, que tingui poc temps per a jugar a un videojoc llarg, i vulgui fer una partida ràpida, per exemple en el cafè del matí.
El jugador ha d'esperar una experiencia simple i ràpida, pero que requereix que el jugador hagi de pensar i planificar. El jugador ha de sentir una emoció de victoria al crear una torre amb característiques elevades i un sentiment de poder. Alhora, ha de sentir tensió mentre cada ronda es fa més i més complexa.

3. Planificació

El projecte es va iniciar el Desembre de 2022. Després dels mesos de Desembre i Gener de crear el prototip inicial i rebre l'aprovació del tutor, el projecte es va posar en marxa.

3.1. Pla de Treball

El projecte es divideix en les següents tasques:

- Pluja d'idees: Generar una idea novedosa sobre uns conceptes saturats. Mecàniques complexes i simples d'escalar per a una complexitat en el cas d'un videojoc complert.
- Implementació del personatge: El disseny del personatge no es completarà en aquests projecte, però si les mecàniques. Definir mecàniques de vida, moviment i energia.
- Crear el mapa i recursos procedurals: Estudiar mètodes procedurals de creació de nivells i aplicar-los al projecte per afegir un nivell de aleatorietat en cada partida.
- Disseny i implementació d'enemics simples: Estudiar un sistema aleatori de creació d'enemics on apareguin segons la ronda on es troba el jugador. Aquest projecte no s'enfoca en la intel·ligència artificial dels enemics, així que estarà molt simplificada.
- Implementació d'un inventari simple: Dissenyar i desenvolupar una base de dades on el jugador podrá obtenir i utilitzar recursos. Tant en la creació d'estructures com altres objectes.
- Disseny i implementació de Torres: Dissenyar i implementar una estructura "Torre" que pugui detectar, apuntar i disparar a objectes que siguin considerats com un objectiu i estiguin en una àrea propera.
- Disseny i implementació del sistema de millores: Aquesta és la tasca que requerirà més temps del projecte i es tracta de dissenyar i implementar una mecànica complexa de millora d'objectes. La mecànica ha de ser interactiva i fácil d'entendre però difícil de dominar. S'han de dissenyar i implementar tant les mecàniques com els objectes que el jugador haurà d'interactuar per utilitzar-la.

- Testeig i arreglar errors: Comprovar que totes les parts funcionen correctament, i afegir o anotar possibles millores que s'haurien d'aplicar al projecte.
- Implementar UI i menús: Dissenyar i implementar la interfície d'usuari i els menús del joc.
- Implementar música i efectes de so: Dissenyar i implementar música i efectes de so simples.

TEMPS	Desembre	Gener	Febrer	Març	Abril	Maig	Juny
Pluja d'idees							
Mecàniques del jugador							
Mapa i recursos							
Torres i enemics							
Cicle nit i dia							
Sistema de millores							
Testeig d'errors							

Figura 2.2.12. Diagrama de Gantt

3.2. Metodologia

El projecte està pensat per generar una base sólida per a un videojoc final. Per tant, el disseny i la implementació dels aspectes del projecte han de tenir les següents parts:

- Escalable: S'ha de poder ampliar o reduir mecàniques segons la necessitat del projecte de la manera més senzilla possible.
- Compatible: S'ha de poder utilitzar en altres projectes o parts del projecte si escau.
- Simple: Reduir la complexitat per tal de maximitzar l'eficiència a l'hora d'ampliar el projecte.

3.3. Resultats esperats

El resultat final del projecte és un prototip jugable de la idea del projecte amb totes les parts mencionades anteriorment afegides o preparades per ser afegides en un futur.

4. Marc del treball

En aquest capítol s'explicarà el marc de treball del projecte. Tant el tipus de videojoc i els referents del projecte com les eines utilitzades i la raó d'utilitzar-les.

4.1. Referències

Hi han molts gèneres i tipus de videojocs en el mercat. Des de shooters realistes a plataformes 2D d'estil cartoon. En aquest projecte s'ha decidit combinar dos generes: Rogue-Like i Tower defense.
Aquesta combinació ha estat utilitzada en varies obres, però aquest projecte vol portar la idea al 3D i afegir una mecànica original. A continuació, s'introduiran els principals referents i les similituds i diferències.

4.1.1 Bloons TD 6

Figura 4.1.1. Logotip de Bloons TD 6

Bloons TD 6 és un videojoc Tower defense que es pot jugar en mòvil o ordinador. El jugador haura de defensar una base d'uns globus que volen destruir la utilitzant torres i altres defenses. Cada torre podrà ser millorada fins a 5 cops en 3 arbres de millora diferents. El videojoc s'enfoca en la gestió de recursos, sinergies entre torres i debilitats de l'enemic.

La diferencia principal entre bloons i aquest projecte és que bloons és en 2D, mentre que el projecte serà en 3D. A més, bloons té un únic recurs, les monedes.

4.1.2 Rogue Towers

Figura 4.1.2. Logotip de Rogue Towers

Com s'ha explicat anteriorment, Rogue Towers és un Rogue lite enfocat en la aleatorietat. El joc és en 2D isomètric i hi ha una gran importància en les millores permanents després de cada partida.

4.1.3 Muck

Figura 4.1.3. Logotip de Muck

A diferencia dels videojocs explicats anteriorment, Muck és un videojoc de supervivencia enfocat en el crafteig i recol·lecció d'objectes i millores.
La diferencia principal és que Muck manca d'un sistema amb estructures defensives i el jugador s'ha de defensar únicament amb armes.

4.1.4 Risk of Rain 2

Figura 4.1.4. Logotip de Risk of Rain 2

Risk of Rain és un Rogue Lite de supervivencia enfocat en l'habilitat del jugador i objectes que el jugador pot trobar pels diferents nivells. El jugador pot elegir entre 12 personatges diferents, cadascú amb les seves pròpies habilitats i estil de joc. Com a diferència principal al projecte, Risk of rain 2 s'enfoca en la velocitat i habilitat del jugador i manca un sistema de recol·lecció i creació d'objectes.

4.2 Motor de videojoc

La creació dels videojocs requereixen un entorn de desenvolupament anomenat *motor de videojocs*. Aquests ofereixen un entorn on poder construir videojocs per a consoles o altres sistemes. Els motors de videojocs ofereixen diverses funcionalitats bàsiques on s'inclou un motor de renderitzat de gràfics 2D i 3D, un motor de càlcul de física , un sistema de detecció i resposta de col·lisions, sistemas de so, programació,animació, gestió de recursos, etc.

4.2.1 Unity 3D

Figura 4.2. Logotip de Unity

Com a motor, el projecte serà realitzat en Unity3D, que és un motor de videojocs gratuït sempre que el any fiscal no superi els 100.000 USD. S'ha elegit aquest motor per les següents raons:

- És un motor fàcil d'utilitzar en equips petits: La alternativa a Unity més propera, Unreal Engine, està especialitzada en equips grans i ofereix recursos específics a aquests. Unity proporciona eines fàcils d'utilitzar per equips tant grans com petits.
- Programació en C#: Les alternatives utilitzen el seu llenguatge propi; blueprints en Unreal Engine i GdScript en Godot. Un dels objectius del projecte era estudiar la generació procedural, i s'ha considerat que la millor manera d'estudiar el subjecte era utilitzar programació convencional.
- Experiència previa: Unity3D és el motor que l'autor té més experiència utilitzant.

4.3. Eina de disseny 3D

Els videojocs estan formats per una gran quantitat d'objectes amb la seva propia geometria. En els videojocs 3D cal generar aquesta geometria utilitzant eines de disseny 3D. Les eines de disseny 3D ofereixen un entorn que permet generar geometria, manipular polígons i punts de la geometria, generar materials, manipular, generar UV i exportar geometria i materials a altres eines o motors. A continuació explicarem la que hem fet servir.

4.3.1 Blender

Figura 4.3. Logotip de Blender

Com a eina de disseny, el projecte utilitzarà blender. Blender és una eina de disseny 3D gratuita, ofereix una gran quantitat de serveis i la possibilitat d'afegir més fent ús d'addons:

- És un motor fàcil d'utilitzar i té una comunitat activa: Blender ofereix els seus serveis d'una manera simple i una gran llibreria de tutorials i ajudes per a principiants.
- Gratuït: Les alternatives a blender, Maya i Houdini requereixen pagar per poder utilitzar els seus serveis en un projecte. Blender ofereix la funcionalitat completa de manera gratuita.

4.4. Equip de treball

Aquest projecte serà realitzat individualment per l'autor del projecte. És l'encarregat de programar, crear models 3D, arreglar errors, etc,

4.5. Conceptes Previs

A l'hora de crear un videojoc hi existeix una nomenclatura i varis entorns amb diversos conceptes que cal conèixer per crear un projecte i pot provocar dificultat en la lectura. En aquest apartat explicarem els conceptes que s'utilitzaràn, juntament amb descripcions i imatges per que es pugui continuar llegint amb facilitat.

4.5.1. Vocabulari anglès utilitzat

- Spawn: Un objecte fa spawn quan s'instancia en l'escenari en temps real.
- Trigger: En l'àmbit de simulacions 3D, un "Trigger" significa que un objecte activar una propietat quan un objecte que tingui una col·lisió entri en contacte amb aquest.
- Mesh: Un conjunt de poligons, punts i linies, unides per formar la forma d'un objecte 3D.
- Noise: Noise és refereix a una funció que retorna números aleatoris, normalment entre 0 i 1.
- Cooldown: Paraula que s'utilitza per representar el temps que necesita un objecte o acció per poder realitzar se. Per exemple si un atac té un cooldown de 3 segons, un cop el jugador utilitza aquest atac haurà d'esperar 3 segons per tornar a utilitzar el mateix atac.

- NPC: Acronim anglès “Non Player Character”. Un NPC és qualsevol personatge o enemic d’un videojoc que no sigui controlat per un jugador.

4.5.2. Editor

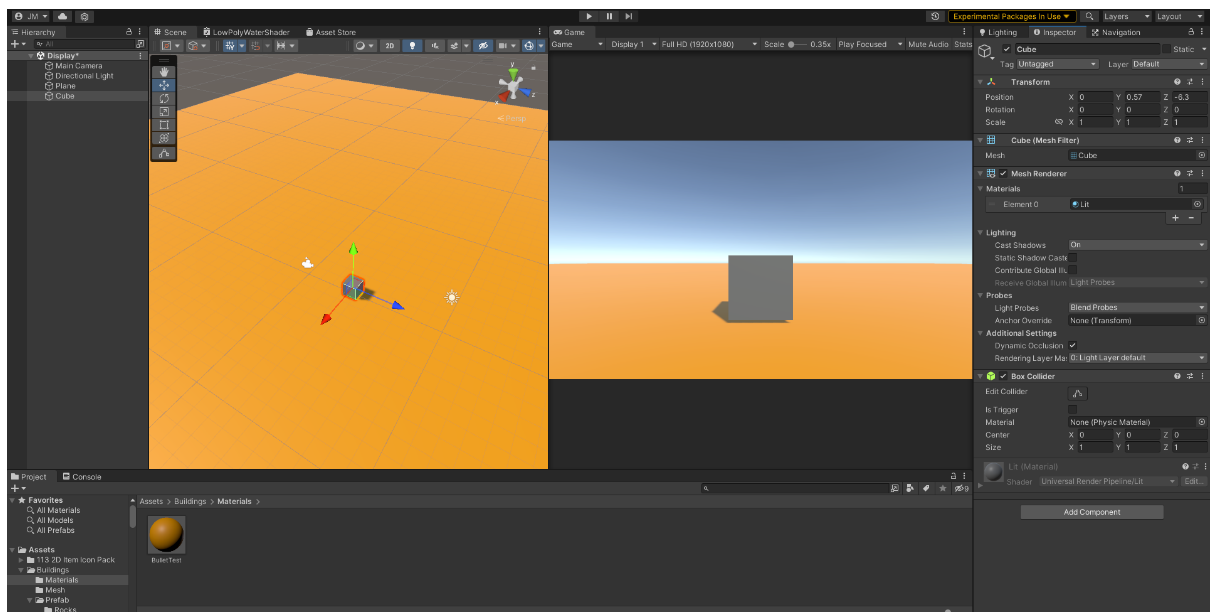

Figura 4.4.1 Captura de pantalla del editor

L’editor és l’espai on un desenvolupador treballa en els projectes. (Veure Figura 4.4.1) En l’editor el desenvolupador pot afegir objectes, mirar i editar la informació d’objectes, consultar les carpetes del projecte, simular el funcionament del videojoc, etc. (Veure figura 4.4.2)

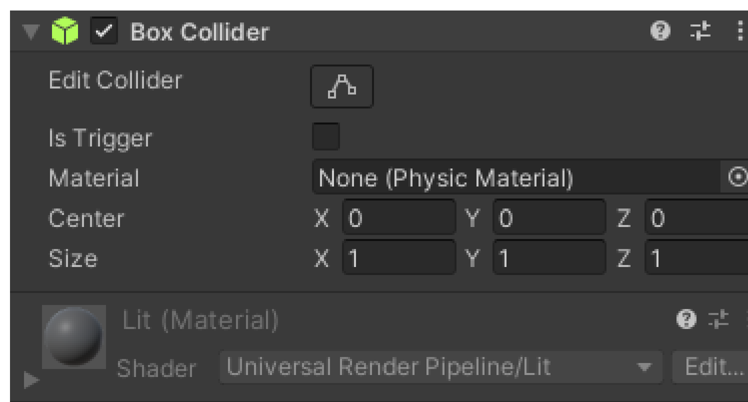

Figura 4.4.2. Component d’un objecte de l’editor

Hi han conceptes explicats més endavant que mencionen l'editor, especificament si un objecte esta actiu en l'editor o no quan la simulació esta pausada (Veure Figures 4.4.3; 4.4.4)

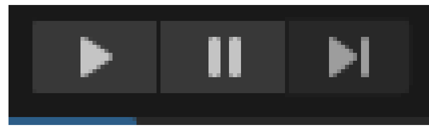
Figura 4.4.3. Simulació inactiva

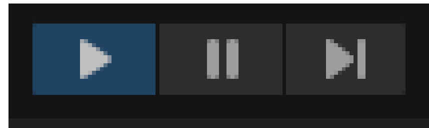
Figura 4.4.4. Simulació Activa

4.5.3. Scripts

Un script és un component que normalment s'afegeix a un objecte de unity per tal que actuï d'una manera específica. Els scripts utilitzen C# i hereten el comportament d'un objecte de Unity anomenat *Mono Behaviour,* que permet que aquest script funcioni dins de la simulació i proporciona funcions com *Start i Update.* (Veure Figura 4.4.5)

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

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

    }

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

    }
}
```

Figura 4.4.5. Script buit de unity

Hi ha alguns scripts que no utilitzen mono behaviour, és a dir, no s'afegeixen a un objecte.

4.5.4. Col·lisions i triggers

Per a què un objecte pugui interactuar amb altres, cal que tinguin una propietat anomenada Collider (Veure Figura 4.4.6)

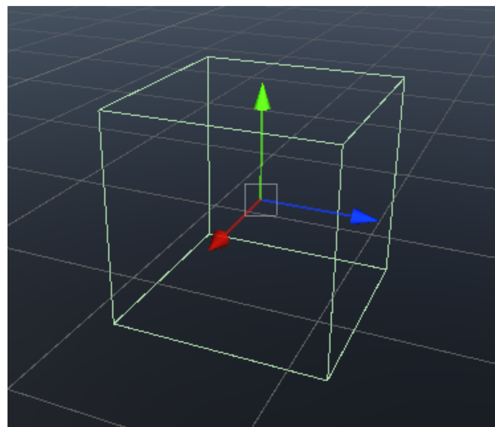

Figura 4.4.6. Collider d'un cub

Aquest collider, si pot interactuar físicament amb objectes, per exemple el terra o el jugador, podem dir que té col·lisió. Un trigger és similar a un objecte amb collisions, però aquest no interactua amb altres objectes. Un trigger activa una funció quan entra en contacte amb un objecte. Per exemple un projectil amb trigger pot activar una funció al tocar un enemic.

4.5.5. Raycast i SphereCast

Raycast és una operació on s'envien raigs invisibles en una direcció concreta per observar si ha tocat algun objecte d'interès. (Veure Figura 4.4.7)

Figura 4.4.7. Visualització de raycast

Aquesta operació és utilitzada en molts àmbits de videojocs, per exemple en shooters s'utilitza per detectar si una bala impactarà a un enemic. També es pot detectar si un jugador toca el terra utilitzant un raycast apuntant cap abaix. En aquest projecte ens interessa la variant spherecast, una alternativa al raycast per detectar objectes en una àrea. El funcionament és similar a raycast, però detecta objectes dins d'una esfera. Estudiarem aquesta variant més a fons quan expliquem el funcionament de la detecció d'enemics.

4.5.6. Sistema d'esdeveniments

En un videojoc, és inevitable que el jugador, o el escenari, canviï o fagi accions que afectin l'experiencia. Aquestes accions s'anomenen esdeveniments.
Els esdeveniments es poden utilitzar de diverses formes, per exemple a l'activar un interruptor una porta s'obre o quan un jugador agafa un objecte aquest s'afegeix a l'inventari.
En aquest projecte els events s'utilitzaran majoritàriament per canvis en l'estat del jugador i gestió d'inventari.

4.5.7. Mesh i Materials

Cada objecte 3D està format per un Mesh Renderer, i un Mesh filter.
El Mesh filter indica quina geometría és la de l'objecte i el mesh renderer especifica els materials i com interactua l'objecte amb la il·luminació. (Veure Figura 4.4.8)

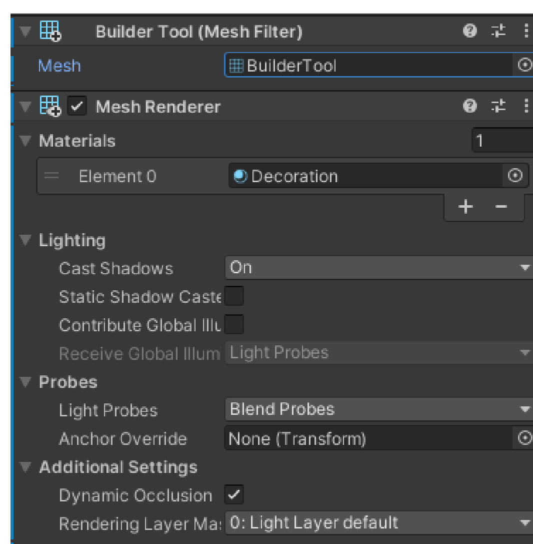

Figura 4.4.8. Dades del Mesh

La geometria és una sèrie de polígons junts formant una forma específica. Cada polígon té diferents dades que especifiquen com es veurà l'objecte final, la normal, l'ordre dels vertex, etc. El material d'un objecte indica les propietats dels polígons d'aquest objecte: el color, la posició, la transparència, com afecta la llum, etc.

4.5.7. Nav Mesh

Nav mesh és un sistema de Unity que s'utilitza en el pathfinding d'NPCs. Es basa en utilitzar la geometria del nivell i els canvis d'alçada per generar un espai on els diferents NPCs es poden moure. (Veure Figura 4.4.9)

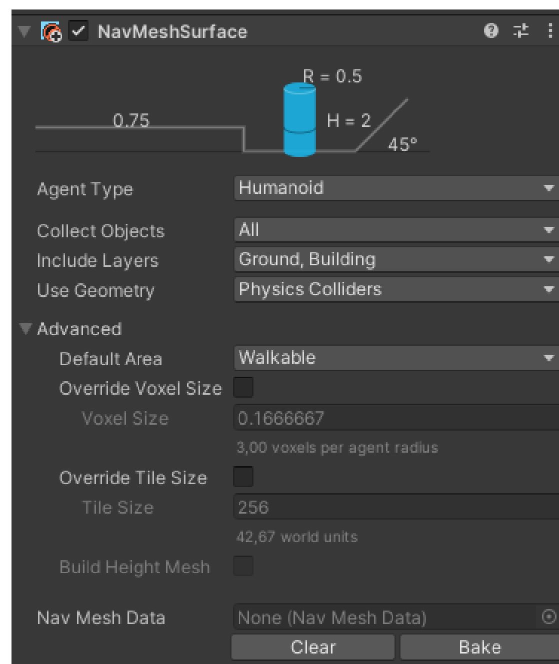

Figura 4.4.9. Dades del Nav Mesh

Cal especificar si els diferents NPCs poden volar, saltar, etc. En aquest projecte, els enemics només poden caminar.

4.5.8. Prefabs

Un Prefab és un component especial de unity que permet guardar un Objecte configurat per poder ser reutilitzats en el projecte sense haver de tornar a configurarlos. Es poden utilitzar en una gran quantitat de situacions: fer spawn d'enemics, construcció d'escenaris de manera ràpida, organització de l'editor, crear diverses còpies del mateix objecte amb lleugeres modificacions, etc.

4.5.9. Canvas i interficies

Un canvas és un objecte de Unity que permet afegir objectes que pertanyen a la interfície com imatges, text, etc. Poden formar part de l'escenari (cartells, pantalles, etc) o part de la càmera (Menús, HUD, etc). (Veure Figura 4.4.9)

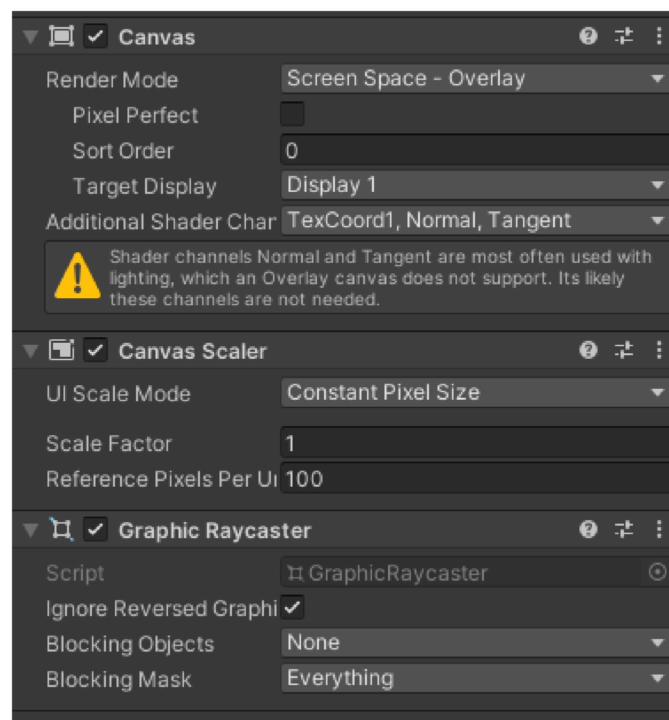

Figura 4.4.10. Dades del Canvas

4.5.10. Scriptable object

Un scriptable object és un objecte de unity que permet crear múltiples instàncies d'un script amb variables úniques. Això permet crear una gran varietat d'objectes amb un rendiment molt alt i poc espai de memoria. Es pot utilitzar per exemple en recursos, enemics, cartes, etc. (Veure Figura 4.4.11)

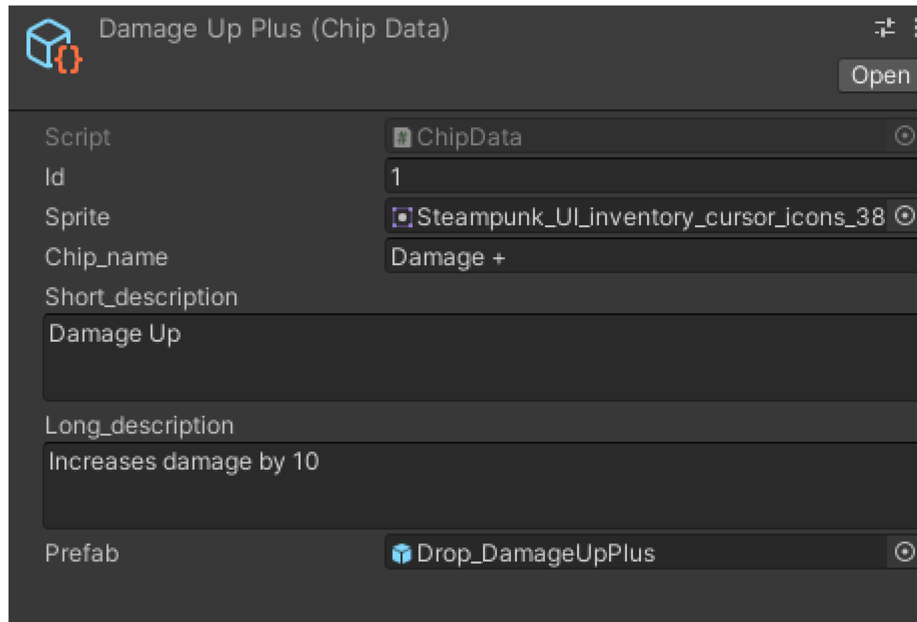

Figura 4.4.11. Exemple d'un scriptable object

5. Disseny del videojoc

En aquest apartat veurem tant els objectius del projecte com els passos necessaris que cal dur a terme per assolir els objectius. Les decisions que s'han pres i el motiu. S'inclou aqui les tècniques procedurals, el sistema de spawn d'enemics, el disseny d'objectes, etc

5.1. Objectius de disseny

En aquest projecte es vol desenvolupar un videojoc amb els següents objectius:

- Experiencia curta i única en cada partida.
- Mecàniques interessants que permetin al jugador experimentar amb les eines del joc.
- Sistemes que incitin i fomentin la creativitat i estratègia en temps real.
- Experimentar amb una combinació de gèneres, supervivència i tower defense, poc explorada.

5.2. Disseny del mapa

El videojoc té un únic escenari, una illa generada proceduralment on el jugador podrà interactuar amb objectes i enemics. (Veure Figura 5.1)

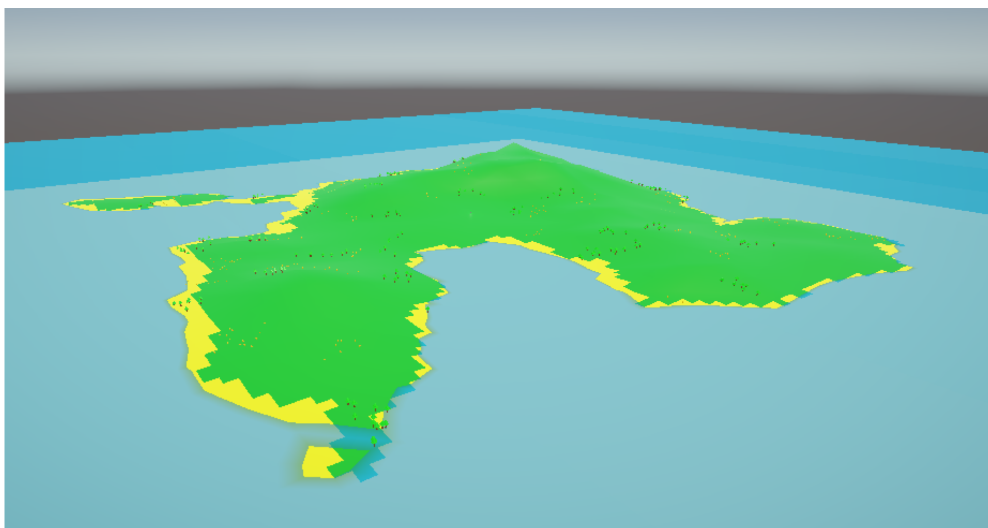

Figura 5.1. Vista aèria de la illa

5.3. Mecàniques

En un videojoc, el jugador ha de poder interactuar amb l'escenari de diverses maneres depenent del gènere de videojoc i objectiu d'aquest. Aquestes interaccions s'anomenen mecàniques.

5.3.1. Accions del personatge

Una de les parts més importants dels videojocs són les accions que el personatge del jugador pot realitzar. Poden ser des-de moure's, saltar fins a disparar una arma o manipular objectes de l'entorn. A continuació veurem les accions que el jugador podrà fer en el projecte.

5.3.1.1 Energia

Similar a la vida real, quan una persona realitza accions, es cansarà. Per tant s'ha implementat un sistema d'energia. Al realitzar accions que requereixin esforç com atacar o correr, el jugador gasta energia. Al no fer ninguna acció en un temps especific, el jugador comença a regenerar aquesta.
Aquesta mecànica ajuda a donar importància altres mecàniques com correr, que en casos on no hi ha un sistema d'energia, aquesta acció perd pes. Quan el jugador realitza una acció, aquest perd energia (Veure figura 5.2.1)

Figura 5.2.1. Jugador perd energia

Quan el jugador no fa cap acció recupera energia lentament (Veure Figura 5.1.2)

Figura 5.2.2 Jugador recupera energia

5.3.1.2. Moviment

A l’hora de moure’s per l’escenari, el jugador pot realitzar diverses accions: caminar, correr i saltar. El jugador es pot moure en qualsevol direcció. Caminar no utilitza energia però el jugador anirà lent. En casos on el jugador es trobi en una situació amb enemics ha de considerar caminar i estalviar energia o consumir energia i correr. El jugador també pot saltar sense utilitzar energia, però en aquest projecte saltar no s’utilitzarà, o no tindrà usos més que decoració.

5.3.1.2. Atac i recol·lecció

El jugador tindrà una eina en l’inventari que li permetrà atacar utilitzant energia. L’atac li permetrà destruir objectes de l’entorn o eliminar enemics. Quan el jugador destrueix un objecte, aquest obtindrà recursos. Per exemple, al destruir un arbre obtindrà fusta que podrà utilitzar en altres mecàniques.

5.3.1.3. Construcció.

Al obtenir els recursos necessaris, el jugador podrà utilitzar una eina per a construir estructures defensives per atacar enemics. Aquestes estructures es podran millorar.

5.4. Gestió de recursos

El jugador podrà obtenir diferents recursos en la partida. Alguns d’aquests recursos s’obtenen d’objectes de l’entorn i altres s’obtenen d’enemics. Els

recursos tenen la funció de crear altres objectes o millores que ajudin al jugador a sobreviure. (Veure Figura 5.3)

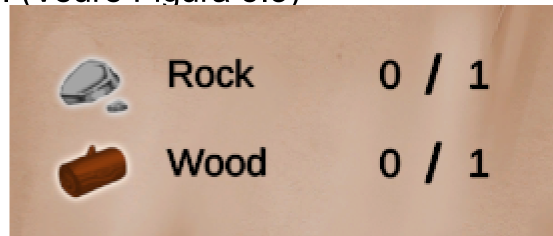

Figura 5.3 Materials requerits per un objecte

Els recursos poden utilitzar-se de les següents formes:

- Construir estructures defensives per a protegir al jugador.
- Construir "chips" que s'utilitzen en el sistema de millora de les estructures defensives.

5.5. Cicle nit i dia

El videojoc estarà dividit en dues parts: nit i dia. En el dia el jugador podrà recolectar recursos i construir defenses de manera segura i en la nit el jugador s'haurà de defensar de rondes d'enemics. Cada cicle serà més difícil amb enemics més poderosos. (Veure Figura 5.4.1; 5.4.2)

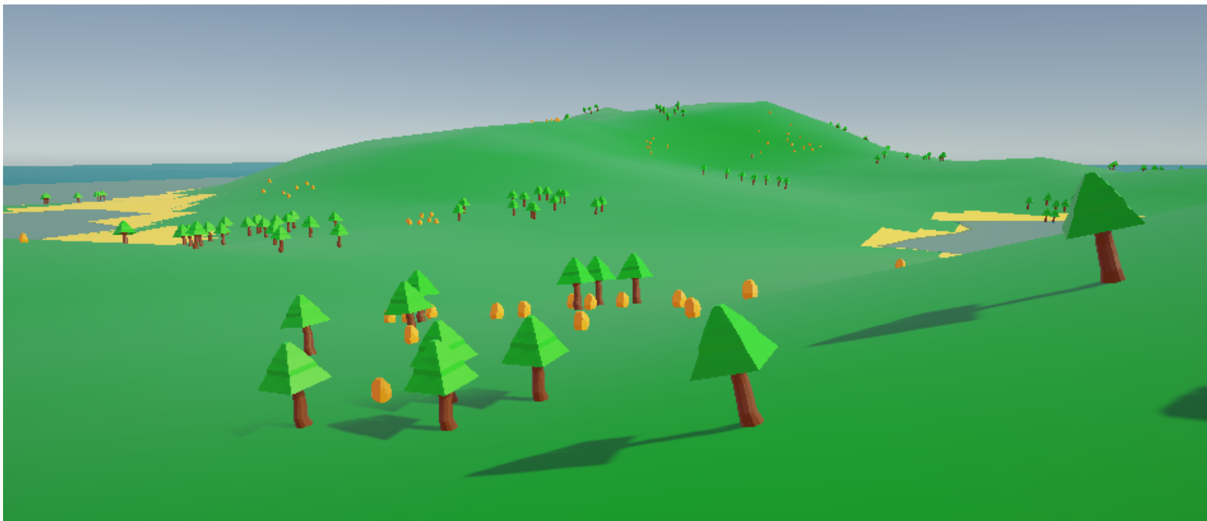

Figura 5.4.1 Imatge del videojoc de dia

Figura 5.4.2 Imatge del videojoc de nit

5.6.Menú

Una part essencial per a un videojoc és el menú. La majoria de videojocs tenen els seguents:

- Menú principal: El menu de l'inici del joc on podrà elegir la configuració del joc, sortir de l'aplicació o iniciar el joc. A vegades hi han altres opcions com tutorials o credits.
- Menú de pausa: El menú que el jugador veurà al prémer el botó de pausa. La funcionalitat d'aquest menú depèn del videojoc. En alguns videojocs és similar a les opcions del menú principal, en altres permet accions com veure l'inventari, veure un mapa, etc (Veure Figura 5.5.1)

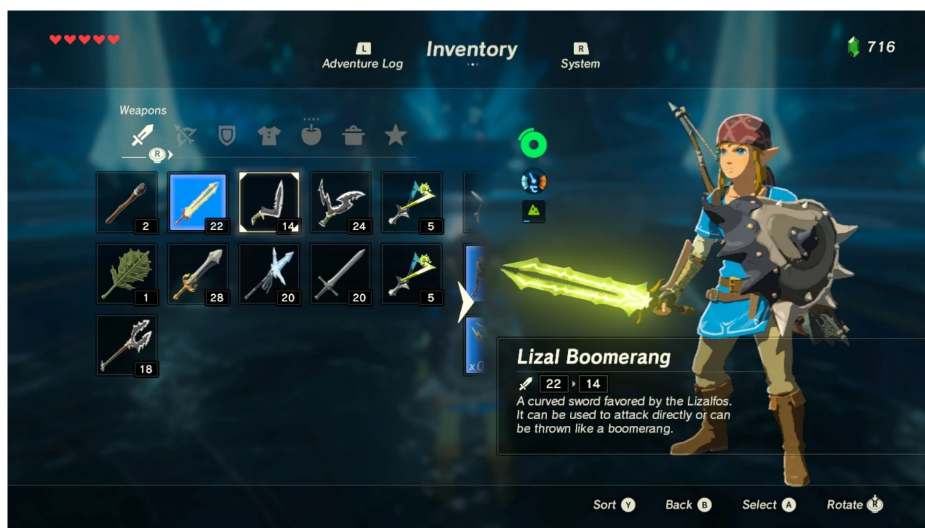

Figura 5.5.1 Menú de pausa de The Legend Of Zelda: Breath of the wild

5.6.1. Menú principal

El menú principal del projecte tindrà les següents funcions:
- Iniciar el joc
- Modificar el volum del joc
- sortir de l'aplicació

5.6.2 Menú de pausa

El menú de pausa del projecte tindrà les següents funcions:
- Sortir del joc
- Modificar el volum del joc
- Veure un petit menú d'ajuda per les mecàniques del joc (Veure Figura 5.5.2)

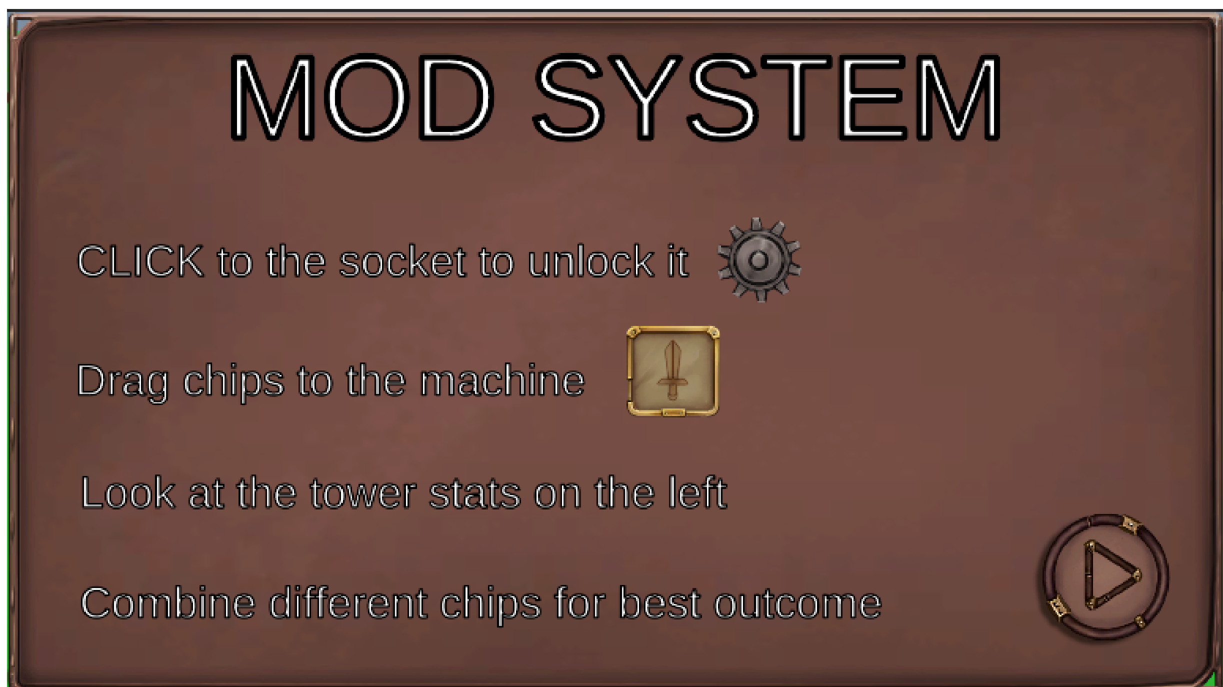

Figura 5.5.2 Captura de pantalla del menu d'ajuda

5.6. Reptes i objectius

L'objectiu principal del joc és sobreviure utilitzant els recursos que el jugador trobarà pel nivell i intentar aguantar el major nombre de rondes, definides pel cicle dia i nit ,on una ronda és un dia sencer de joc o entre 6-10 minuts de temps real. (Veure Figura 5.6)

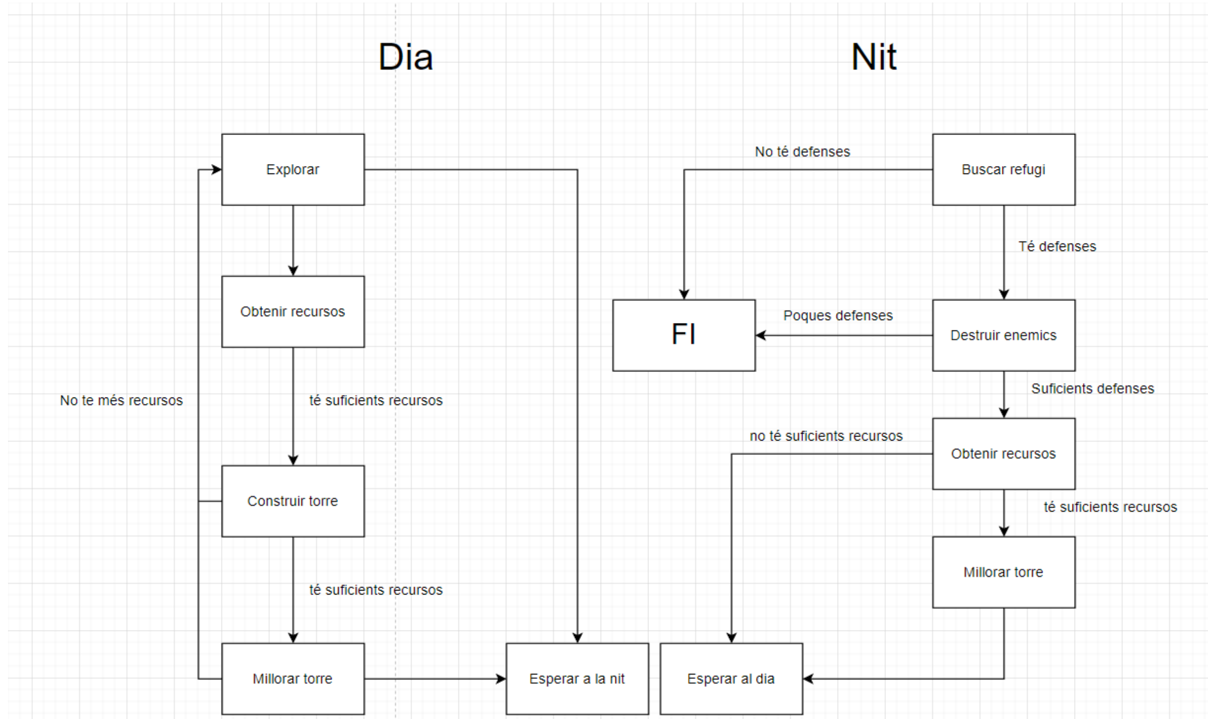

Figura 5.6 Diagrama de flux

5.7. Estètica

La estética d’aquest projecte és low poly amb colors vius. Inspirada principalment en el videojoc Muck. (Veure Figura 5.7)

Figura 5.7 Captura de pantalla de Muck

Com el jugador haurà de construir defenses i millores, les interfícies tindran una estética “Steampunk” similar a jocs com Bioshock o Dishonored. (Veure Figura 5.2.7;5.2.8)

5.7.1 Models 3D

El projecte és un videojoc 3D i, per tant, requereix models d’objectes 3D en l’escenari. El videojoc requereix els models de les eines del jugador; un pic i una eina de construcció, els enemics i la torre de defensa amb la consola de millores. (Veure Figures 5.8.1 a 5.8.5)

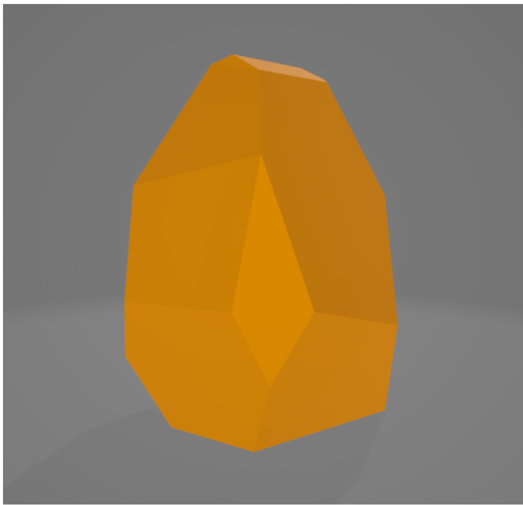

Figura 5.8.1 Model d’una pedra

Figura 5.8.2 Model d’un arbre

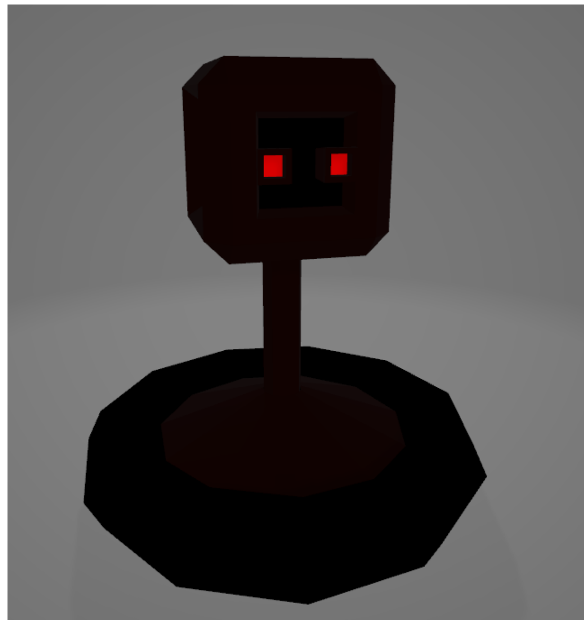

Figura 5.8.3 Model d’un robot

Figura 5.8.4 Model d’un pic

Figura 5.8.5 Model de l’eina del jugador

5.7.2 Interfície

Pell disseny de l’interficie del videojoc s’utilitzarà un dels paquets que proporciona Gentleland en la *Asset Store* de Unity 3D.

Figura 5.9.1 Logotip de Gentleland

Gentleland ofereix un paquet de recursos gratuïts d’UI d'estètica steampunk que utilitzarem pel disseny de les interfícies. (Veure Figura 5.9.2)

Figura 5.9.2 Icones Steampunk que ofereix Gentleland

A més del disseny dels menús, també necesitarem icones pels diferents recursos. Utilitzarem el paquet d'icones que ofereix gratuitament *Mimu Studio.* per les icones dels diferents objectes: fusta, pedra, etc. (Veure Figura 5.9.3)

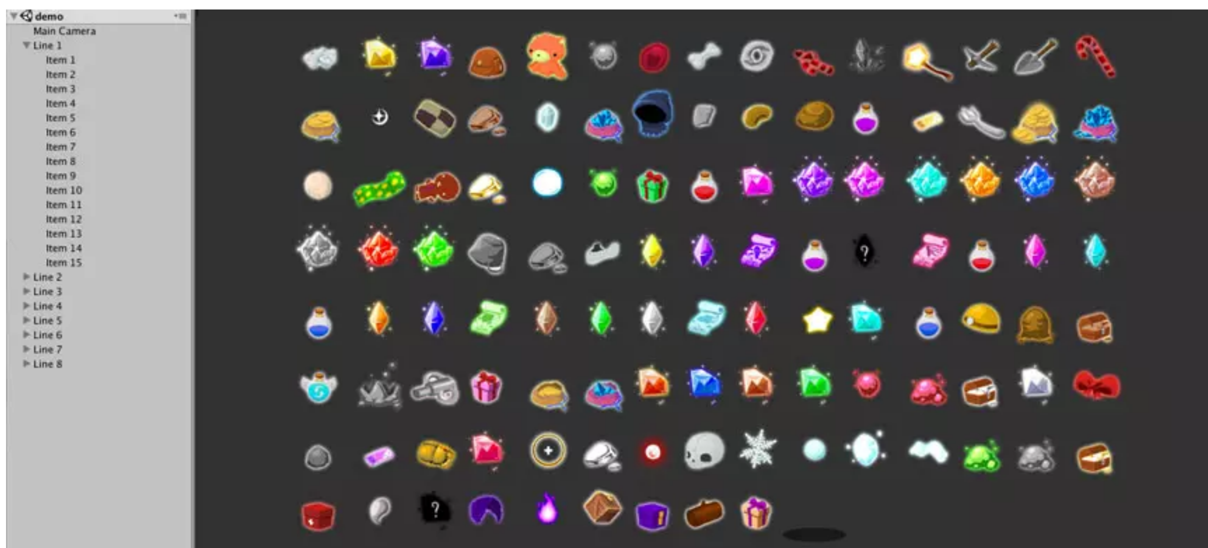

Figura 5.9.3 Imatge de mostra de Mimu Studio

Cal tenir en compte que no s'utilitzaran totes les icones que ofereix tan enGentleland com Mimu Studio, només aquells que requereixi el projecte.

5.8. Elements a desenvolupar

En aquest apartat, mostrarem els elements que haurem de desenvolupar en aquest projecte. Es dividiran en diferents apartats segons les diferents parts del videojoc.

5.8.1 Menú

- Menú Principal: Implementar menú principal.
- Menú d'ajuda: Implementar un menú d'ajuda.
- Menú de Pausa: Implementar el menú de pausa.
- Menú de fi de la partida: Implementar el menú del fi de la partida.

5.8.1 HUD

- Barra de vida: Implementar una barra de vida.
- Barra d'energia: Implementar una barra d'energia.
- Receptes: Implementar interfície de receptes.
- Eines: Implementar interficie d'eines.

5.8.2 Models 3D

- Enemics: Crear i implementar models d'enemics
- Torre: Crear i implementar el model de la torre.
- Consola de modificació: Crear i implementar tan la interfície com el model de la consola de modificació.
- Recursos: Crear i implementar models de recursos. Tant la versió completa com la destruida.

5.8.3.Generació procedural

- Generació procedural del nivell: Implementar un nivell creat proceduralment.
- Recursos aleatoris: Crear i implementar una funció que crei recursos en posicions aleatòries del nivell.
- Generació aleatoria d'enemics: Crear i implementar un sistema aleatori de generació d'enemics.

5.8.4.Cicle nit i dia

- Visual: Crear i implementar un cicle nit i dia.
- Mecànica: Crear i implementar un sistema de rondes que segueixi el cicle nit i dia.

5.8.5. Mecàniques

- Personatge: Implementar les mecàniques del personatge del jugador.
- Sistema de defensa: Implementar les mecàniques de defensa d'enemics.
- Sistema de millora: Implementar les mecàniques de millora de les defenses.

5.9. Objectius descartats o per a un futur

A causa d'un enfoc més ampli a les mecàniques explicades anteriorment i que el projecte es realitza per una persona, s'ha hagut de descartar les següents mecàniques que hauria agradat que formessin part del projecte i que serien afegides en el cas de que el projecte es portes a un videojoc final:

- Varietat d'eines: El jugador pot utilitzar més d'una arma per defensar se.
- Sistema de millores de l'eina: El jugador pot millorar la seva arma.
- Sistema elemental i de debilitats: El sistema de millores afegiria elements a les defenses que afectarien de diferents maneres als enemics.

- Narrativa o objectiu final: El jugador hauria d'obtenir objectes específics per a poder escapar de la illa o aconseguir un objectiu final i guanyar la partida.
- Objectes aleatoris de millora del jugador: El jugador podria trobar objectes que millorin les seves característiques com la vida, l'energia, etc.
- Varietat de estructures defensives: El jugador podria construir muralles i torres amb diferents tipus d'atac.
- Sistema de millores aleatori: Els objectes de millora es trobarien en el món o eliminant enemics i el jugador no tindria totes les opcions des de el principi.
- Sistema de millores fora del joc: El jugador podria aconseguir reptes dins del joc que desbloquejarà nous objectes per a partides futures.

6. Implementació

En aquest apartat explicarem totes les parts del joc i com s'han implementat. Explicarem tant les tècniques que s'han utilitzat com problemes i raons per les quals les utilitzem en el projecte.

6.1 Objectes destructibles

El jugador es troba objectes en el món que haurà de destruir. Per tal d'aconseguir aquest efecte utilitzarem l'script "BreakObject". (Veure Figures 6.1; 6.4; 6.5)

```
Script de Unity (2 referencias de recurso) | 2 referencias
public class BreakObject : MonoBehaviour
{
    [SerializeField] private GameObject destroyedMesh;
    [SerializeField]
    private ResourceDropManager resource_drop;

    [SerializeField] private float max_health;
    [SerializeField] private EnemyHealthDisplay display;

    private float current_health;
```

Figura 6.1 Variables de l'script "BreakObject"

A la figura 6.1 podem veure el codi on afegim una vida a l'objecte que el jugador haurà de destruir utilitzant el pic. A més, hem d'afegir l'objecte "destroyedMesh", que és una variable fragmentada de l'objecte que volem destruir (Veure Figures 6.1; 6.2) i els recursos que obtindrà el jugador quan es destrueix.

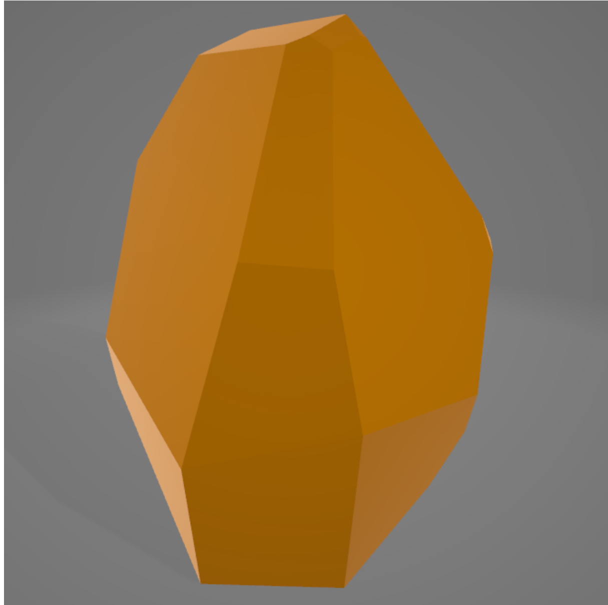

Figura 6.2 Mesh original

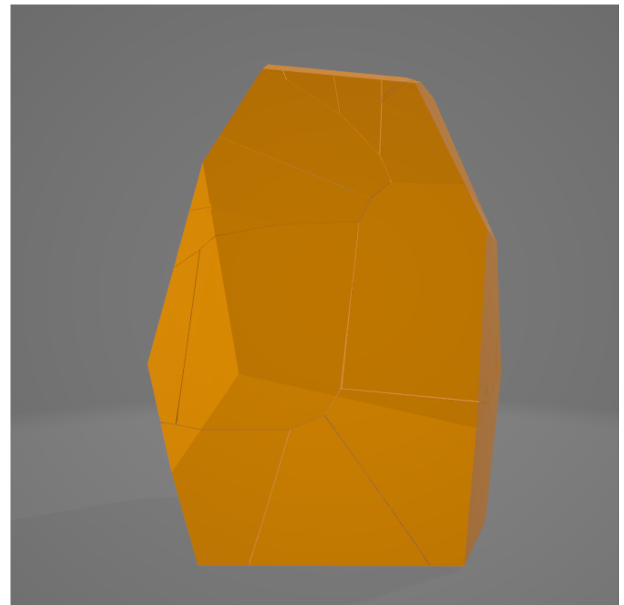

Figura 6.3 Mesh fragmentat

Inicialitzem la vida actual de l'objecte al instanciar (Veure Figura 6.4)

```
Mensaje de Unity | 0 referencias
private void Start()
{
    current_health = max_health;
    display.SetMaxHp(max_health);
}
```

Figura 6.4 Funció "Start" de l'script "BreakObject"

Un cop instanciat les variables l'objecte no fa res fins que el jugador l'ataqui. Llavors cridem la funció "Take_Damage", mostrat a la figura 6.5

```
1 referencia
public void Take_Damage(float damage)
{
    float true_damage = DamageCalculator.CalculateDamage(damage, 0);

    Debug.Log(true_damage);
    current_health -= true_damage;
    display.onTakeDamage(true_damage);
    if (current_health <= 0)
    {

        Instantiate(destroyedMesh, transform.position, transform.rotation);
        resource_drop.SpawnResource();
        Destroy(this.gameObject);
    }
}
```

Figura 6.5 Funció “Take_Damage” de l’script “BreakObject”

La funció Take_Damage calcularà el mal que ha rebut al ser atacat pel jugador i, si aquest és menor o igual que 0, crea l’objecte fragmentat en la seva posició i rotació, donarà recursos al jugador, i s’eliminarà.
L’efecte final és el mostrat a les Figures 6.6 i 6.7.

Figura 6.6 Arbre sense destruir

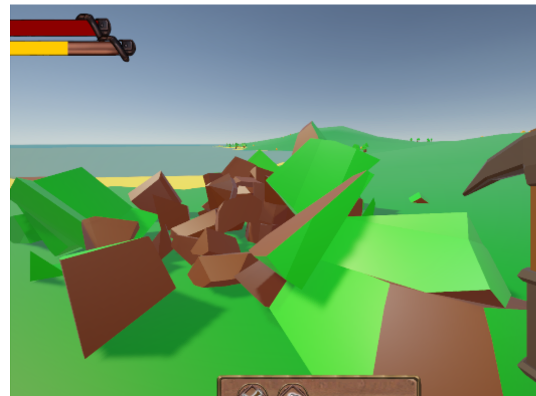

Figura 6.7 Arbre destruït

Cada fragment de l’objecte destruit és un objecte amb físiques i col·lisions, pel que tenir molts provocaria problemes de rendiment en partides llargues. Per tant, utilitzem l’script “DespawnBrokenObject” que té la funció d’eliminar l’objecte al cap d’un temps. (Veure figura 6.8)

```
Script de Unity (3 referencias de recurso) | 0 referencias
public class DespawnBrokenObject : MonoBehaviour
{
    [SerializeField] private float active_time;

    private bool despawning = false;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        StartCoroutine(Wait());
    }
    Mensaje de Unity | 0 referencias
    void Update()
    {
        if (despawning)
        {
            Destroy(this.gameObject);
        }
    }

    1 referencia
    IEnumerator Wait()
    {
        despawning = false;
        yield return new WaitForSeconds(active_time);
        despawning = true;
    }
}
```

Figura 6.7 Script “DespawnBrokenObject”

6.2 Cicle dia i nit

Abans d'explicar el funcionament dels scripts que participen en el cicle nit i dia, cal un coneixement previ sobre el funcionament dels gradients que explicarem a continuació.

6.2.1 Gradients i percentatges

Podem considerar un gradient com una sèrie de valors entre 0 i 1, on l'inici del gradient és 0 i el final 1. (Veure Figura 6.12)

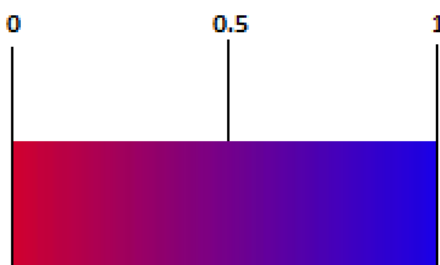

Figura 6.12 Visualització de gradient

Aquesta sèrie de valors ens permet substituir un percentatge pel color corresponent al gradient. En el nostre cas volem substituir el percentatge de temps de cicle en la simulació. Utilitzant l'exemple anterior (Veure Figura 6.12), podem veure que, per exemple, el valor 0.75 dona un color violeta.

6.2.2 Implementació

Per crear un efecte de dia i nit crearem un "scriptable object" que tingui gradients de colors. Aquests colors determinen el color i l'efecte de l'il·luminació (Veure Figures 6.8; 6.9)

```
[System.Serializable]
[CreateAssetMenu(fileName ="Lighting Preset",menuName ="Lighting/Preset")]
Script de Unity | 1 referencia
public class DayCycleLighting : ScriptableObject
{
    public Gradient AmbientColor;
    public Gradient DirectionalColor;
    public Gradient FogColor;

}
```

Figura 6.8 Scriptable Object pel cicle nit i dia

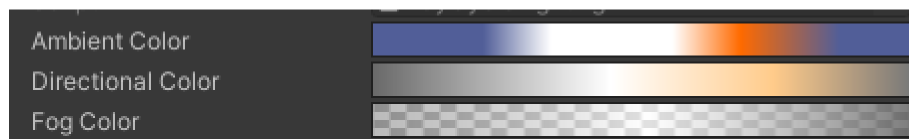

Figura 6.9 Gradients de llum

Per aconseguir un efecte nit i dia crearem l'script "GameCycleManager" que calculara el temps del dia i substituira el color de la font de llum per la variable "DirectionalColor", la llum ambiental per "AmbientColor" i el color de la boira per "Fog Color" segons el percentatge del dia en el que es troba el joc. Al arribar a un percentatge concret, el joc detecta si s'ha fet de dia o de nit i n'informa al sistema d'events (Veure Figures 6.9; 6.10; 6.11)

```
[SerializeField] private Light DirectionalLight;
[SerializeField] private DayCycleLighting Preset;
[SerializeField] private float time_for_cycle;

[SerializeField] private float TimeOfDay;

[SerializeField, Range(0,1)] private float morning_percentage=0.2f;
[SerializeField, Range(0, 1)] private float dawn_percentage=0.8f;

[SerializeField]
public bool auto_update;


private bool day_time = false;
```

Figura 6.9 Variables de l'script "GameCycleManager"

S'inicialitza la font de llum (el sol), les variables mencionades anteriorment, el temps que dura un cicle i el temps actual. A més s'inicialitza el percentatge on

el joc detecta si és nit o dia i si el temps passa automàticament. (Veure Figura 6.9)

```
Mensaje de Unity | 0 referencias
private void Update()
{
        TimeOfDay += Time.deltaTime;
        TimeOfDay %= time_for_cycle;
        UpdateLighting();
}
```

Figura 6.10 Funció Update de l'script "GameCycleManager"

Cada frame de joc, l'script calcula el temps actual i crida la funció que actualitza la il·luminació (Veure Figura 6.10)

```
2 referencias
public void UpdateLighting()
{


    float time_percent = TimeOfDay / time_for_cycle;
    if (!day_time && time_percent>morning_percentage &&
        time_percent < dawn_percentage)
    {
        Debug.Log("DayTime");
        GameEvents.current.MorningTriggerEnter();
        day_time = true;
    }
    if (day_time && time_percent > dawn_percentage)
    {
        GameEvents.current.DawnTriggerEnter();
        Debug.Log("NightTime");
        day_time = false;
    }
    RenderSettings.ambientLight = Preset.AmbientColor.Evaluate(time_percent);
    RenderSettings.fogColor = Preset.FogColor.Evaluate(time_percent);

    if (DirectionalLight != null)
    {
        DirectionalLight.color = Preset.DirectionalColor.Evaluate(time_percent);
        DirectionalLight.transform.localRotation = Quaternion.Euler(new Vector3((time_percent * 360)
            - 90f, 170, 0));
    }
}
```

Figura 6.11 Funció "UpdateLighting" de l'script "GameCycleManager"

En la funció mostrada a la Figura 6.11 es calcula el percentatge actual del dia dividint el temps de cicle entre el temps actual. Si es detecta que és de dia o nit, l'script envia un event indicant que el temps ha canviat. Finalment substitueix la il·luminació amb els colors del gradient, utilitzant el percentatge calculat a l'inici. A més, és rota la font d'il·luminació per generar un efecte similar a la terra girant al voltant del sol.

6.3 Editor

L'editor de unity permet crear scripts que modifiquin o afegeix funcions a l'editor per tal de simplificar o accelerar el procés de creació. En aquest projecte hem creat 2 scripts que s'utilitzen en el cicle dia i nit i en la generació procedural (Veure Figures 6.12; 6.13). Aquests scripts tenen com a única funció fer proves.

```
[CustomEditor(typeof(GameCycleManager))]
Script de Unity | 0 referencias
public class DayCycleEditor : Editor
{
    0 referencias
    public override void OnInspectorGUI()
    {
        GameCycleManager mapgen = (GameCycleManager)target;

        if (DrawDefaultInspector())
        {
            if (mapgen.auto_update)
            {
                mapgen.UpdateLighting();
            }
        }
    }
}
```

Figura 6.12 Script "DayCycleEditor" de l'editor

```
[CustomEditor(typeof(MapGenerator))]

public class MapGeneratorEditor : Editor
{

    public override void OnInspectorGUI()
    {
        MapGenerator mapgen = (MapGenerator)target;

        if (DrawDefaultInspector())
        {
            if (mapgen.auto_update)
            {
                mapgen.generate_map();
            }
        }
        if (GUILayout.Button("Generate Map"))
        {
            mapgen.generate_map();
        }

    }
}
```

Figura 6.13 Script "MapGeneratorEditor" de l'editor

6.4 Enemics

Instanciar i crear enemics aleatoris és un tema complex i s'han utilitzat diverses estratègies al llarg dels anys. En aquest projecte utilitzarem una estrategia desenvolupada per "Hopoo Games" a l'hora de crear "Risk of Rain" anomenada "Directors".

6.4.1 Directors i spawn cards

Un director és un objecte en el món que s'encarrega d'instanciar enemics, objectes, etc. En el nostre projecte els directors es limitaran a instanciar enemics. Un director té un sistema intern de punts que poden gastar per a generar objectes. Aquests punts es poden generar tant en el temps com a l'inici de rondes, segons com sigui necessari. El director, a més de punts, té una llista d'objectes anomenada "Spawn Cards" on hi apareix l'objecte a instanciar, en el nostre cas enemics, el valor en punts i el pes d'aparició. El pes d'aparició determina la possibilitat d'obtenir aquesta carta.
El funcionament és simple. Segons la necessitat del joc, el director elegira a l'atzar un numero de cartes fins que es quedi sense punts o no tingui punts suficients per crear nous objectes. (Veure Figura 6.14) En aquest projecte, s'utilitza una versió simplificada on no hi ha pes de les cartes.

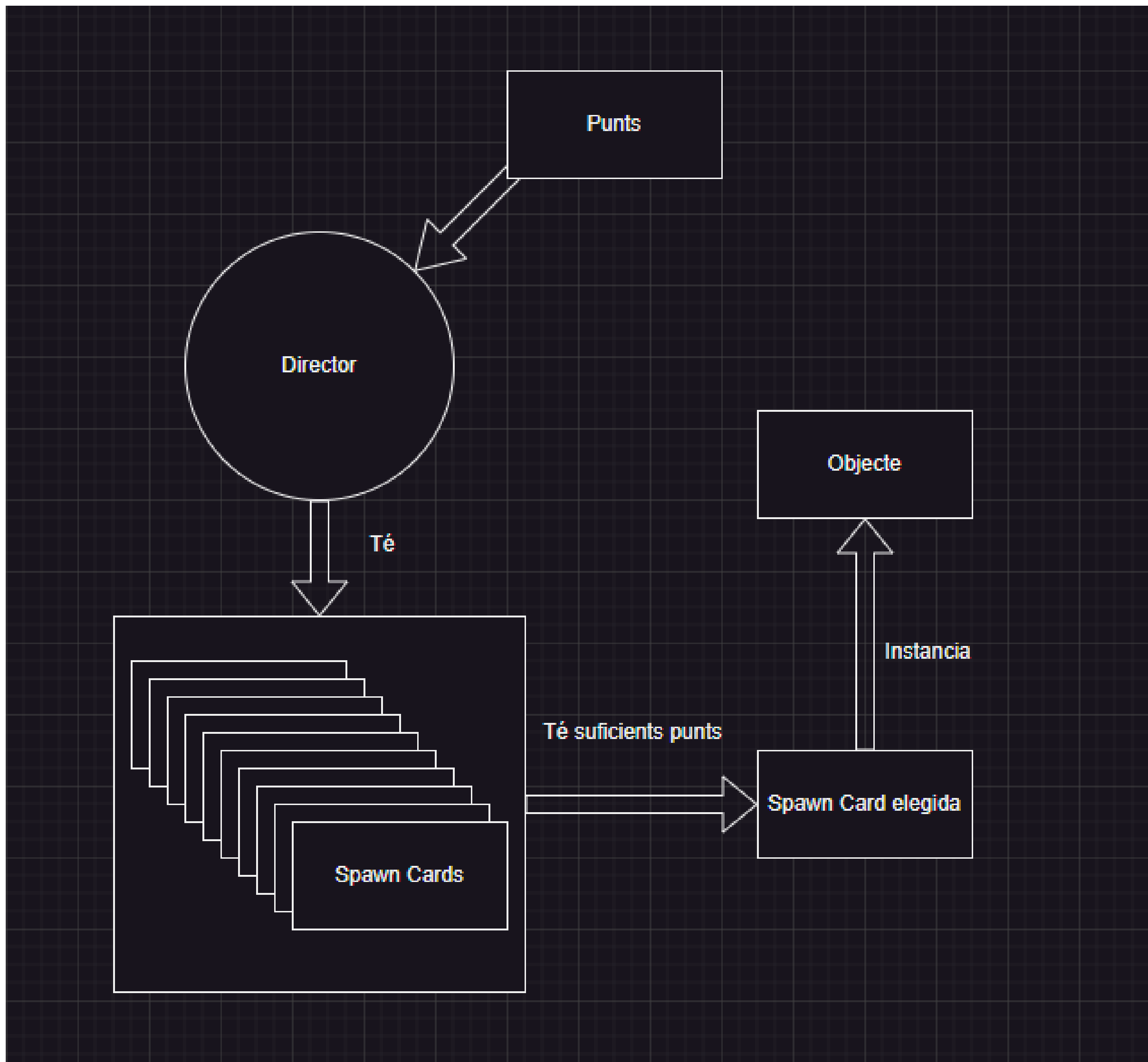

Figura 6.14 Visualització del funcionament del director

6.4.2 Nav Mesh

A l’hora de crear un enemic, és important que hi hagi una àrea on hi pot navegar. Aquesta àrea s’anomena “Nav Mesh” o mesh de navegació. Unity ofereix eines per crear un mesh de navegació (Veure Figura 6.15)

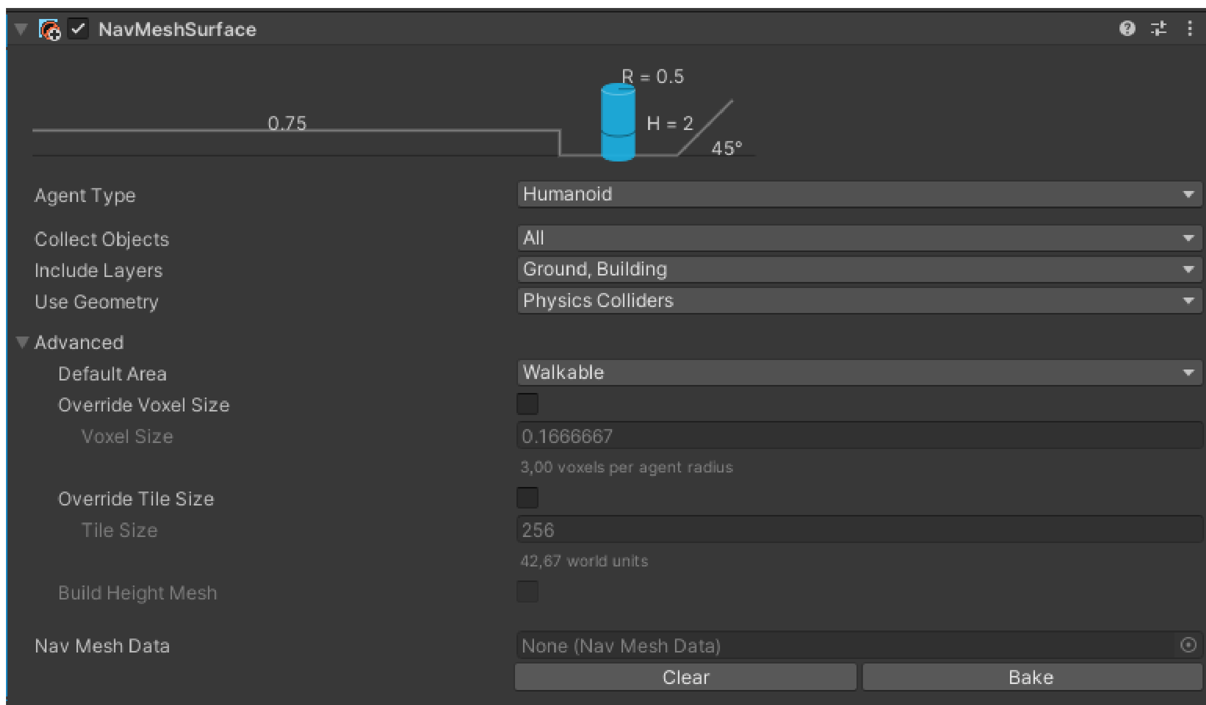

Figura 6.15 Objecte generador de nav mesh

Ja que el nostre escenari és generat proceduralment a l'inici de cada partida, no és possible generar el nav mesh prèviament i estalviar temps de càrrega. Aquest problema ha portat a dues solucions diferents. La primera és crear un nav mesh al voltant del jugador que s'actualitzi quan aquest s'ha mogut una distancia. (Veure Figura 6.16)

```
Mensaje de Unity | 0 referencias
void Update()
{
    if (Vector3.Distance(transform.position, surface.transform.position) >= distanceThreshold)
        UpdatePosition();
}

void UpdatePosition()
{
    surface.transform.position = transform.position;
}
```

Figura 6.16 Primera solució al generar un nav mesh

La segona solució i la finalment elegida és crear un nav mesh a l'inici de cada partida del mapa (Veure Figura 6.17) . Aquesta solució requereix un temps de càrrega al iniciar la partida i gasta més recursos, però provoca menys errors a l'hora de generar enemics, per tant, és la solució que utilitzarem.

```
void UpdateNavMesh()
{
    surface.RemoveData();

    surface.BuildNavMesh();
}
```
Figura 6.17 Funció d’actualització

Cal tenir en compte que la funció que actualitza el nav mesh (Veure Figura 6.17) s’utilitza en ambdues solucions, la diferència és únicament la mida del mesh i les vegades que s’utilitza.

6.4.3 Implementació enemics

Els enemics estan dividits en 3 parts: les característiques, el moviment i el comportament. Per cada part afegirem un script, més un script que servirà per mostrar al jugador la vida de l'enemic. Aquest script també s'utilitzarà en l’apartat de recursos.

Les dades de l’enemic es guardaran en un scriptableobject que guarda tant el tipus d’enemic com variables i l’augment de variables per ronda. (Veure Figura 6.18). Algunes característiques no s’utilitzen en el projecte i serveixen com a placeholder en el cas d’una versió final.

```
[CreateAssetMenu(fileName = "Enemy_Data", menuName = "Data/Enemy", order = 1)]
Script de Unity | 2 referencias
public class EnemyData : ScriptableObject
{


    1 referencia
    public enum enemy_type
    {
        Boss,Miniboss,Basic
    }
    0 referencias
    public enum enemy_movement
    {
        Grounded,Air
    }

    [Header("Descriptions")]
    public string enemy_name;
    [TextArea(minLines:3,maxLines:4)]
    public string enemy_description;
    public string enemy_ID;


    [Header("Stats")]
    public float damage;
    public float health;
    public float speed;
    public float armor;
    public float vision_range;
    public float vision_angle;

    [Header("Scaling")]

    public float health_per_round;
    public float armor_per_round;
    public float speed_per_round;

    [Header("Behaviour")]
    public enemy_type type_enemy;
}
```

Figura 6.18 Scriptableobject “EnemyData”

Pel comportament de l’enemic utilitzarem l’script “EnemyHandler. Aquest tindrà vida i armadura que augmenta depenent de la ronda actual. Al rebre mal per part d’un jugador i perdre tota la vida, aquest morirá i donarà recursos al jugador. (Veure Figures 6.19; 6.20)

```
[SerializeField] private EnemyData data;
[SerializeField] private EnemyHealthDisplay display;

[SerializeField]
private ResourceDropManager resource_drop;

private float max_health;
private float max_armor;

private float current_health;
private float current_armor;

private int current_round = 1;

// Start is called before the first frame update
Mensaje de Unity | 0 referencias
void Start()
{
    max_health = data.health + data.health_per_round * current_round;
    max_armor = data.armor + data.armor_per_round * current_round;

    current_health = max_health;
    current_armor = max_armor;

    display.SetMaxHp(max_health);

}
```

Figura 6.19 Inicialització de l'script "Enemy Handler"

```
2 referencias
public void Take_Damage(float damage)
{
    float true_damage = DamageCalculator.CalculateDamage(damage,current_armor);

    Debug.Log(true_damage);
    current_health -= true_damage;
    display.onTakeDamage(true_damage);
    if (current_health <= 0)
    {
        if (resource_drop != null)
        {
            resource_drop.SpawnResource();
        }

        Destroy(this.gameObject);
    }
}
Mensaje de Unity | 0 referencias
private void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.tag == "Player")
    {
        collision.gameObject.GetComponent<PlayerStateHandler>().Take_Damage(data.damage);
    }
}
```

Figura 6.20 Detecció de col·lisions i atacs de l'script "EnemyHandler"

Per calcular el dany que rep l'enemic en un atac, utilitzem la funció de la Figura 6.21

```
3 referencias
public static float CalculateDamage(float damage,float current_armor)
{
    float true_damage = damage - damage * current_armor / (Mathf.Abs(current_armor) + 100);
    true_damage = Mathf.Round(true_damage * 10) / 10;
    return true_damage;
}
```

Figura 6.21 Funció de càlcul de dany

Aquesta funció asegura que el enemic rebi més dany amb armadura negativa y menys amb armadura positiva, sense reduir el dany a 0.

Finalment, el moviment està dividit en dues parts: Detecció de l'objectiu i moviment cap a l'objectiu. Primer inicialitzem les dades de l'enemic per veure, l'objectiu, l'àrea de visió i l'agent de navegació que farà ús del mesh de navegació (Veure Figura 6.22). Cal tenir en compte que en el projecte final l'àrea de l'enemic és global i que, per tant, sempre anirà a la seva direcció. La visió local es pot utilitzar en cas d'una versió final amb objectius de baixa prioritat si l'enemic no detecta al jugador o no pot arribar a ell, com torres o altres estructures.

```
[RequireComponent(typeof(NavMeshAgent))]
Script de Unity (4 referencias de recurso) | 2 referencias
public class EnemyMovement : MonoBehaviour
{
    [SerializeField] private EnemyData data;
    [SerializeField]
    LayerMask player_layer;
    private NavMeshAgent navigation_agent;

    private float speed;
    private int current_round=1;


    private GameObject player_target;
    private bool can_see_player;
    private float normal_vision;
    [SerializeField] private bool global_range = false;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Start()
    {
        normal_vision = data.vision_angle;
        navigation_agent = GetComponent<NavMeshAgent>();
        speed = data.speed + data.speed_per_round*current_round;
        player_target = GameObject.FindGameObjectWithTag("Player");
        StartCoroutine(FOVRoutine());


        navigation_agent.speed = speed;
    }
```

Figura 6.22 Inicialització de l'script “EnemyMovement”

La visió del enemic pot cambiar de local a global utilitzant la funció de la Figura 6.23

```
public void VisionRangeMode(bool mode)
{
    global_range = mode;
}
```

Figura 6.23 Funció de tipus de visió de l'script "EnemyMovement"

Per tal d'estalviar recursos, utilitzarem una rutina que detecti enemics cada 0.2 segons en comptes de cada frame. Per detectar objectes es genera una esfera al voltant de l'enemic de la mida de la seva visió o del mapa si la visió es global, i es comprova si hi ha algun objecte del tag a buscar, en aquest cas el jugador. Si existeix, el enemic pot veure al jugador. (Veure Figura 6.24)

```
1 referencia
private void handleDetection()
{
    float detection_range = normal_vision;
    if (global_range)
    {
        detection_range = 10000f;
    }
    Collider[] hitColliders = Physics.OverlapSphere(transform.position, detection_range, player_layer);
    if (hitColliders.Length != 0)
    {
        can_see_player=true;
    }
    else if (can_see_player)
    {
        can_see_player = false;
    }

}
1 referencia
private IEnumerator FOVRoutine()
{
    WaitForSeconds wait = new WaitForSeconds(0.2f);

    while (true)
    {
        yield return wait;
        handleDetection();
    }
}
```

Figura 6.24 Detecció del jugador de l'script "EnemyMovement"

Pel moviment, a cada frame l'enemic elegeix l'objectiu. Si veu al jugador l'elegeix com a objectiu, si no el veu, no es mou (Veure Figura 6.25). El moviment el realitza el component "Nav Mesh Agent" (Veure Figura 6.26).

```
1 referencia
private void handle_movement()
{
    if (navigation_agent.isOnNavMesh)
    {
        if (can_see_player)
        {
            NavMeshPath path = new NavMeshPath();
            navigation_agent.CalculatePath(player_target.transform.position, path);
            if (path.status != NavMeshPathStatus.PathPartial)
            {

                navigation_agent.SetDestination(player_target.transform.position);
            }

        }
        else
        {
            navigation_agent.SetDestination(transform.position);
        }
    }
}
```

Figura 6.25 Moviment de l'script "EnemyMovement"

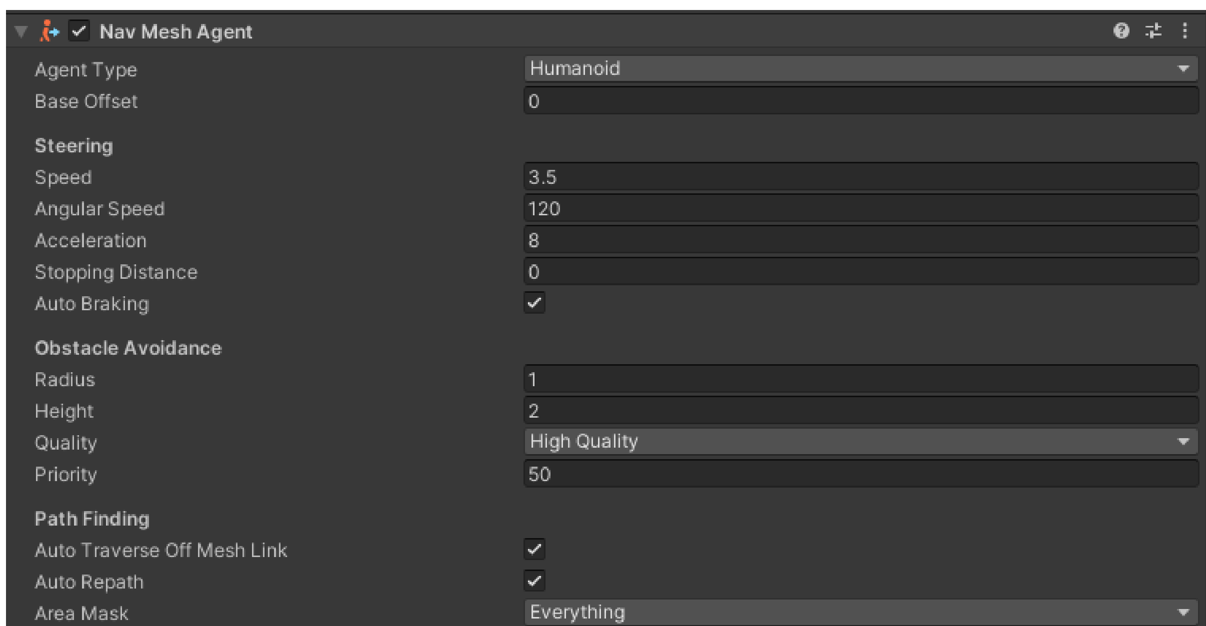

Figura 6.26 Component "Nav Mesh Agent"

6.4.4 Gestió de rondes

Per gestionar les rondes utilitzem un objecte que té l'script "RoundHandler". Aquest script té una llista de directors que s'encarregaran de crear enemics, la ronda actual i l'estat (si és nit o dia) i els crèdits que rebrà cada director a l'inici de la ronda. A més, és subscriurà al sistema d'esdeveniments per saber si la ronda és activa o no (Veure Figura 6.27)

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class RoundHandler : MonoBehaviour
{
    [SerializeField] private int current_round = 0;

    [SerializeField] private int starting_enemy_credits;
    [SerializeField] private int credits_per_round;

    [SerializeField] private Spawner[] spawners;

    private bool roundActive = false;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        GameEvents.current.onMorningEnter += StopRound;
        GameEvents.current.onDawnEnter += StartRound;
    }
```

Figura 6.27 Inicialització de l'script "RoundHandler"

Aquest script té 3 funcions: Iniciar una nova ronda, parar la ronda actual i activar o desactivar els directors. (Veure Figura 6.28)

```
1 referencia
public void StartRound()
{
    roundActive = true;
    current_round++;
    int current_credits = starting_enemy_credits + credits_per_round * current_round;

    AssignSpawnerValues(current_round, roundActive, starting_enemy_credits);
}
1 referencia
public void StopRound()
{
    roundActive = false;
    AssignSpawnerValues(current_round, roundActive);
}
2 referencias
private void AssignSpawnerValues(int current_round,bool round_state,int extra_credits=0 )
{
    for (int i = 0; i < spawners.Length; i++)
    {
        spawners[i].AssignRound(round_state, current_round);
        if (extra_credits>0)
        {
            spawners[i].IncreaseCredits(extra_credits);
        }
    }
}
```

Figura 6.28 Funcions de l’script “RoundHandler”

6.4.5 Implementació director

Com hem explicat anteriorment utilitzarem un sistema simplificat de directors. Utilitzem l’script “Spawner” per aconseguir aquesta funció i scriptable objects que anomenem“SpawnCards”. Les cartes que utilitzem són simples. Tenen el cost d’ús, la ronda on es poden utilitzar i l'enemic que es crea al utilitzar-se. (Veure Figura 6.29)

```
[CreateAssetMenu(fileName = "Spawn_Card", menuName = "Data/Spawn_Card", order = 1)]
Script de Unity | 1 referencia
public class SpawnCard : ScriptableObject
{
    public int spawn_cost;
    public GameObject entity;
    public int round_possible;
}
```

Figura 6.29 Scriptable object “SpawnCard”

El director té com a variables els crèdits actuals, la ronda actual, els crèdits que obté en el temps, una llista de cartes, l'àrea on crea enemics, el temps que tarda en crear enemics i l'estat de la ronda. (Veure Figura 6.30)

```
private int currency;
private int round=0;

[SerializeField]
private int round_passive_currency;

[SerializeField] private int max_iterations;
[SerializeField] private SpawnCard[] entities;

[SerializeField] private float range_of_spawn;

[SerializeField]
private float time_per_object_slow = 5;
[SerializeField]
private float time_per_object_fast = 1;

[SerializeField] private bool round_is_active=false;
private bool canSpawn = true;
private bool generating_credits = false;
```

Figura 6.30 Inicialització de l'script "Spawner"

Per generar una carta, el director explora les cartes que té aleatòriament un numero especificat de vegades i, quan troba una carta possible, la genera i resta els crèdits corresponents. (Veure Figura 6.31)

```
1 referencia
public GameObject GeneratePossibleCard()
{
    int iterations = 0;
    while(iterations<max_iterations)
    {
        int pickedValue= Random.Range(0, entities.Length-1);
        if(entities[pickedValue].round_possible<= round && currency>= entities[pickedValue].spawn_cost)
        {
            currency -= entities[pickedValue].spawn_cost;
            return entities[pickedValue].entity;
        }
        iterations++;
    }
    return null;
}
```

Figura 6.31 Funció de generar cartes de l'script "Spawner"

Per generar crèdits el director utilitza una rutina que genera credits depenent de la ronda cada segon. (Veure Figura 6.32)

```
2 referencias
public void IncreaseCredits(int credits)
{
    currency += credits;
    Debug.Log(currency);
}
1 referencia
IEnumerator GenerateCredits()
{
    generating_credits = true;

    IncreaseCredits(round_passive_currency*round);
    yield return new WaitForSeconds(1f);

    generating_credits = false;
}
```

Figura 6.32 Generació de crèdits de l'script "Spawner"

Finalment, per generar enemics, el director crea una rutina on genera enemics i els instancia en una posició aleatoria dins del rang d'influència. (Veure Figura 6.33)

```
2 referencias
IEnumerator Spawn(float time)
{
    canSpawn = false;
    Vector3 random_position = new Vector3(Random.Range(transform.position.x - range_of_spawn, transform.position.x + range_of_spawn),
        transform.position.y , Random.Range(transform.position.z - range_of_spawn, transform.position.z + range_of_spawn));
    GameObject prefab = GeneratePossibleCard();
    if (prefab != null)
    {
        GameObject entity = Instantiate(prefab, random_position, Quaternion.identity);
        if (entity.GetComponent<EnemyMovement>() != null)
        {
            entity.GetComponent<EnemyMovement>().VisionRangeMode(round_is_active);
        }
    }
    else
    {
        Debug.Log("No enemy spawned");
    }
    yield return new WaitForSeconds(time);
    canSpawn = true;
}
```

Figura 6.33 Funció per Instanciar enemics de l'script "Spawner"

La funció de generar enemics (Veure Figura 6.33) depèn del temps. El director assigna aquest temps depenent de l'estat de la ronda actual. (Veure Figura 6.34)

```
// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    if (!generating_credits)
    {
        StartCoroutine(GenerateCredits());
    }
    if (canSpawn)
    {
        if (round_is_active)
        {
            StartCoroutine(Spawn(time_per_object_fast));
        }
        else
        {
            StartCoroutine(Spawn(time_per_object_slow));
        }

    }

}
```

Figura 6.34 Funció Update de l'script "Spawner"

6.4.6 Mostrar la vida actual

Per mostrar la vida del enemic utilitzarem l'script "EnemyHealthDisplay" mostrat a les Figures 6.35 , 6.36 i 6.37. Aquest agafa la vida de l'enemic, la vida actual i la mostra en una barra de vida situada a sobre de l'enemic o objecte quan aquest rep mal. (Veure Figura 6.35)

```
Script de Unity (2 referencias de recurso) | 2 referencias
public class EnemyHealthDisplay : MonoBehaviour
{

    private float max_life;
    private float current_life;
    private float delayed_life;

    [SerializeField] private Image healthBar;
    [SerializeField] private Image Background;
    [SerializeField] private Image delayedBar;


    private bool enemy_injured=false;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Start()
    {
        healthBar.enabled = false;
        Background.enabled = false;
        delayedBar.enabled = false;

        this.GetComponent<Canvas>().worldCamera = Camera.main;

    }
```

Figura 6.35 Inicialització de l'script "EnemyHealthDisplay"

Al rebre mal, l'objecte ha d'actualitzar la barra per veure aquest canvi. Per fer-ho emplenem l'objecte que actua com a indicador de la vida fins al percentatge de vida dividint la vida actual entre la total. (Veure Figura 6.36)

```
2 referencias
public void SetMaxHp(float life)
{
    max_life = life;
    current_life = max_life;
    delayed_life = max_life;
}
2 referencias
public void onTakeDamage(float damage)
{
    current_life = current_life - damage;
    healthBar.fillAmount = current_life / max_life;
    if (current_life<max_life && !enemy_injured)
    {
        enemy_injured = true;
        healthBar.enabled = true;
        Background.enabled = true;
    }
}
}
```

Figura 6.36 Funció per actualitzar la vida l'script "EnemyHealthDisplay"

La barra de vida hauria de seguir la càmera del jugador per no generar problemes a l'hora d'observar la. Per tant, cridem la funció mostrada a la Figura 6.37 cada frame que doni aquesta funció.

```
1 referencia
private void RotateTowardsCamera()
{
    Vector3 v = Camera.main.transform.position - transform.position;
    v.x = v.z = 0.0f;
    transform.LookAt(Camera.main.transform.position - v);
    transform.Rotate(0, 180, 0);
}
```

Figura 6.37 Funció per rotar la càmera de l'script "EnemyHealthDisplay"

6.5 Sistema d’esdeveniments

En un videojoc es pot considerar esdeveniments qualsevol canvi en un objecte de l’escenari. Aquests canvis poden interessar a altres objectes per realitzar accions. Per exemple, quan es prem un botó una porta s’obre. El sistema d’events de unity utiliza una variant del patró de disseny “Observer”.

6.5.1. Observer

El patró “Observer” és un patró de disseny de programació que permet als objectes saber si un esdeveniment s’ha dut a terme. Aquest té dues parts: Un objecte que es observat i notifica canvis i els objectes subscrits. (Veure Figura 6.38)

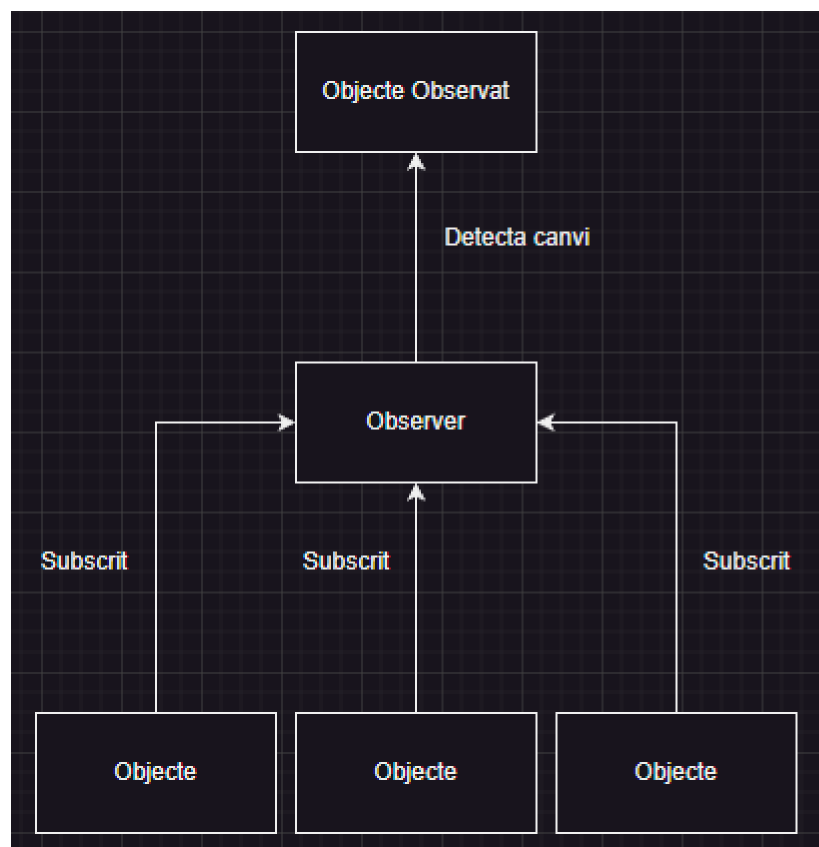

Figura 6.38 Diagrama de flux del patró observer

L’objecte observer detecta canvis en l’objecte i notifica als objectes que estan subscrits. Aquests esperen a la notificació del observer i actuen quan la detecten.
En el cas de Unity, l'objecte observer és un Script que té totes les accions que es consideren com a esdeveniment i l’objecte observat és qui notifica al observer dels canvis cridant la seva funció específica. (Veure Figures 6.39; 6.40)

```
public event Action onMorningEnter;
public event Action onDawnEnter;
```

Figura 6.39 Events de l'objecte observer

```
GameEvents.current.DawnTriggerEnter();
```

Figura 6.40 Objecte observat notificant al observer

6.5.2. Implementació

Utilitzant el sistema d’events de Unity creem l’objecte que funciona com a Observer del projecte i el funcionament en l’script “GameEvents”, on l’objecte observa canvis en els recursos i el temps actual. (Veure Figura 6.41)

```
Script de Unity (1 referencia de recurso) | 10 referencias
public class GameEvents : MonoBehaviour
{
    public static GameEvents current;

    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        current = this;
    }
    public event Action onMorningEnter;
    public event Action onDawnEnter;
    public event Action<ResourceData> onResourceModified;

    1 referencia
    public void MorningTriggerEnter()
    {
        if (onMorningEnter!=null)
        {
            onMorningEnter();
        }
    }
    1 referencia
    public void DawnTriggerEnter()
    {
        if (onDawnEnter != null)
        {
            onDawnEnter();
        }
    }

    3 referencias
    public void ResourceModified(ResourceData data)
    {
        if (onResourceModified != null)
        {
            onResourceModified(data);
        }
    }
}
```

Figura 6.41 Script “GameEvents”

6.6 Handler

Un “handler” és un objecte que té la funció d’ocuparse de les funcions i sistemes del joc que no requereixen un objecte físic per funcionar. Per exemple la generació d’enemics, el cicle nit i dia, gestió de recursos, etc. Això permet una implementació més simple. A continuació explicarem els scripts que utilitzem per gestionar la partida.

6.6.1 Control del cursor

En videojocs 3D en primera persona, és important saber quan s’ha de permetre moure el ratolí, per exemple en menús, i quan no. L’script “Cursor Handler” permet a altres scripts i funcions modificar l’estat de la càmera i retornar aquest estat si es requereix. (Veure Figura 6.42)

```
7 referencias
public static class CursorHandler
{
    public static bool camera_enabled;
    3 referencias
    public static void LockCursor()
    {
        Cursor.lockState = CursorLockMode.Locked;
        camera_enabled = true;
    }
    3 referencias
    public static void UnlockCursor()
    {
        Cursor.lockState = CursorLockMode.None;
        camera_enabled = false;
    }
}
```

Figura 6.42 Script “CursorHandler”

6.6.2 Estat de la partida

En un videojoc el jugador passa per diferents estats: viu, mort, en un menú, etc, i cada estat requereix la seva interficie i propietats. L'script "GameStateHandler" (Veure Figura 6.43; 6.44) s'encarrega de pausar i reiniciar el joc, d'informar si el jugador entra o surt d'un menú o si el jugador ha mort.

```
public class GameStateHandler : MonoBehaviour
{
    public bool game_is_active = true;
    public GameObject game_over_screen;
    public GameObject hud;
    public GameObject menu;
    public GameMenuHandler menuHandler;

    private bool playerOnMenu = false;
    private bool playerOnPaused = false;
    Mensaje de Unity | 0 referencias
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (playerOnPaused)
            {
                PlayerUnpauses();
            }
            else
            {
                PlayerPauses();
            }
        }
    }
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        CursorHandler.LockCursor();
    }
```

Figura 6.43 Inicialització de variables de l'script "GameStateHandler"

```
1 referencia
public void FinishGame()
{
    game_is_active = false;

    game_over_screen.SetActive(true);
    CursorHandler.UnlockCursor();

}

1 referencia
public void PlayerOnMenu()
{
    game_is_active = true;
    hud.SetActive(false);
    CursorHandler.UnlockCursor();
    playerOnMenu = true;

}
1 referencia
public void PlayerLeftMenu()
{
    game_is_active = false;
    hud.SetActive(true);
    CursorHandler.LockCursor();
    playerOnMenu = false;
}
1 referencia
public void PlayerPauses()
{
    playerOnPaused = true;

    game_is_active = false;
    menu.SetActive(true);
    CursorHandler.UnlockCursor();
    playerOnPaused = true;

}
2 referencias
public void PlayerUnpauses()
{
    playerOnPaused = true;

    game_is_active = true;
    menu.SetActive(false);
    CursorHandler.LockCursor();
    playerOnPaused = false;
    menuHandler.CloseMenu();
}
}
```

Figura 6.44 Funcions de l’script “GameStateHandler”

6.7 HUD

El HUD permet al jugador veure informació del estat de la partida. L'script "HUDBarHandler" permet al jugador veure la vida i energia actual en pantalla fent ús d'un control lliscant. (Veure Figura 6.45)

```
Script de Unity | 2 referencias
public class HUDBarHandler : MonoBehaviour
{
    [SerializeField] private Slider slider;

    7 referencias
    public void SetCurrentValue(float value)
    {
        slider.value = value;
    }
    2 referencias
    public void SetMaxValue(float value)
    {
        slider.maxValue = value;
    }

}
```

Figura 6.45 Script "GameOverScreen"

Resource UI display permet mostrar un recurs en pantalla durant uns segons quan el jugador l'obté o el perd. (Veure Figura 6.46)

```
public class ResourceUIDisplay : MonoBehaviour
{

    [SerializeField]
    private ResourceManager manager;

    public bool visible = false;
    public float timer;
    public TextMeshProUGUI text;
    private ResourceData current_data;
    // Start is called before the first frame update

    private void Awake()
    {
        manager = FindObjectOfType<ResourceManager>();
    }


    private void Start()
    {
        GameEvents.current.onResourceModified += ModifyDisplay;
    }

    private void Update()
    {
        if (visible)
        {
            text.text = current_data.name + " " + manager.ResourceAmount(current_data);
        }
        else if(text.text!="")
        {
            text.text = "";
        }
    }

    public void ModifyDisplay(ResourceData data)
    {
        current_data = data;
        StartCoroutine(StartCooldown());
    }

    public IEnumerator StartCooldown()
    {
        visible = true;
        yield return new WaitForSeconds(timer);
        visible = false;
    }
}
```

Figura 6.46 Script “ResourceUIDisplay”

6.8 Objectes interactius

Objectes interactius són objectes que el jugador troba o crea en el mapa amb els que pot interactuar, normalment prement un botó. Hi han varies maneres de crear la interacció. Les més comunes són generar un trigger al voltant d'un objecte i si el jugador és a dins del trigger pot interactuar o, el cas que hem utilitzat, generar un raycast en la direcció on apunta el jugador i ,si apunta a un objecte interactuable en una distancia escollida, el pot utilitzar.
L'objecte interactiu té la frase que apareix en pantalla quan aquest es pot activar, la distancia d'activació i l'efecte al activarse a més de les funcions que permeten a altres scripts obtenir aquestes variables. (Veure Figura 6.47)

```
Script de Unity (1 referencia de recurso) | 5 referencias
public class Interactable : MonoBehaviour
{
    public string interactableText = "Press E to Interact";

    [SerializeField] private GameObject menu_camera;
    public float minDistance = 5f;
    private GameStateHandler handler;
    private GameObject main_camera;
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        main_camera = Camera.main.gameObject;
        handler = FindObjectOfType<GameStateHandler>();
    }
    2 referencias
    public virtual void activateEffect()
    {
            if (handler.game_is_active)
            {
                handler.PlayerLeftMenu();
                main_camera.SetActive(true);
                menu_camera.SetActive(false);
            }
            else
            {
                handler.PlayerOnMenu();
            menu_camera.SetActive(true);
            main_camera.SetActive(false);

        }
    }

    1 referencia
    public bool canInteract(float distance)
    {
        return distance <= minDistance;
    }

    1 referencia
    public string getText()
    {
        return interactableText;
    }
}
```

Figura 6.47 Script "Interactable"

Per tal de comprovar si el jugador pot interactuar amb l'objecte generem un raycast en direcció on apunta el jugador com mostra la Figura 6.48. Si la distància és inferior a la màxima, text apareixerà en pantalla i el jugador podrà prémer el botó "E" per activar l'objecte (Veure Figura 6.49)

```
Debug.DrawRay(Camera.main.transform.position, Camera.main.transform.forward * 200f, Color.yellow);
if (!Physics.Raycast(Camera.main.transform.position, Camera.main.transform.forward, out var hitInfo))
{
    return;
}
if ((bool)hitInfo.collider.gameObject.GetComponent<Interactable>())
```

Figura 6.48 Raycast de l'script "InteractivityManager"

```
if ((bool)hitInfo.collider.gameObject.GetComponent<Interactable>())
{
    if (hitInfo.collider.gameObject.GetComponent<Interactable>().canInteract(hitInfo.distance))
    {
        Debug.DrawRay(base.transform.position, Camera.main.transform.forward * 200f, Color.green);
        Debug.Log("Did Hit");
        if (!textShowing)
        {
            interactText.text = hitInfo.collider.gameObject.GetComponent<Interactable>().getText();
            textShowing = true;
        }
        if (Input.GetKeyDown(KeyCode.E))
        {
            hitInfo.collider.gameObject.GetComponent<Interactable>().activateEffect();
        }
    }
}
else if (textShowing)
{
    interactText.text = "";
    textShowing = false;
}
```

Figura 6.49 Funcionament de l'script "InteractivityManager"

6.9 Navegació de menús

A l'hora de navegar per menús, és útil fer ús del sistema de botons que ofereix unity. En cada botó s'assigna una funció que s'activa quan el jugador el prem. Aquests botons serviran per activar o desactivar menús actuals o per canviar d'escena, en aquest cas del menú principal al joc i viceversa.

6.9.1. Menú principal

Creem l'script "MainMenuButtonHandler" ,mostrat a les Figures 6.50, 6.51 i 6.52, i inicialitzem els objectes en els que el jugador pot navegar: el menú principal, el menú d'opcions, el menú de millores, el menú de sortida que avisa al jugador de si vol sortir del joc o no i el menú de càrrega. (Veure Figura 6.50)

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class MainMenuButtonHandler : MonoBehaviour
{
    [SerializeField] private string GameScene;

    [SerializeField] private GameObject main_menu;
    [SerializeField] private GameObject unlocks_menu;
    [SerializeField] private GameObject options_menu;
    [SerializeField] private GameObject exit_menu;

    private GameObject active_menu;


    //Loading Part
    [SerializeField] private GameObject loading_screen;
    [SerializeField] private Image load_scene_fill;
```

Figura 6.50 Inicialització de l'script "MainMenuButtonHandler

Llavors afegim les funcions que utilitzen els botons del menú, mostrades a la figura 6.51

```
0 referencias
public void UnlocksMenu()
{
    active_menu.SetActive(false);
    unlocks_menu.SetActive(true);
    active_menu = unlocks_menu;
}
0 referencias
public void OptionsMenu()
{
    active_menu.SetActive(false);
    options_menu.SetActive(true);
    active_menu = options_menu;
}

0 referencias
public void ExitGameUI()
{
    exit_menu.SetActive(true);
    active_menu = exit_menu;
}
0 referencias
public void ReturnToMenu()
{
    active_menu.SetActive(false);
    main_menu.SetActive(true);
    active_menu = main_menu;
}
0 referencias
public void ExitGame()
{
    Application.Quit();
}
```

Figura 6.51 Funcions dels botons de l'script "MainMenuButtonHandler

Pel botó d'inici del joc, hem decidit crear una pantalla de càrrega. Per ferla utilitzem una funció asincrona que mostri el el percentatge carregat en una barra, mostrada a la figura 6.52.

```
0 referencias
public void StartGame()
{
    GameSeed.GenerateSeed();
    StartCoroutine(LoadSceneAsync());
}
1 referencia
IEnumerator LoadSceneAsync()
{
    AsyncOperation operation = SceneManager.LoadSceneAsync(GameScene);

    loading_screen.SetActive(true);
    while (!operation.isDone)
    {
        float progress_value = Mathf.Clamp01(operation.progress/0.9f);

        load_scene_fill.fillAmount = progress_value;

        yield return null;
    }
}
```

Figura 6.52 Funció d'inici del joc "MainMenuButtonHandler

6.9.2. Dins de la partida

L'script "GameOverScreen" genera una pantalla de fi del joc amb una frase aleatoria extreta d'una llista de frases quan s'activa. (Veure Figura 6.53)

```
Script de Unity | -referencias
public class GameOverScreen : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI Text;

    [SerializeField] private string[] GameOverTexts;

    Mensaje de Unity | -referencias
    private void OnEnable()
    {
        int position = Random.Range(0, GameOverTexts.Length - 1);
        Text.text = GameOverTexts[position];
    }
}
```

Figura 6.53 Script "GameOverScreen"

L'script "ReturnToMenú" té la funció del botó del menú de pausa per tornar a la pantalla principal (Veure Figura 6.54)

```
Script de Unity | 0 referencias
public class ReturnToMenu : MonoBehaviour
{
    0 referencias
    public void ReturnMenu()
    {
        SceneManager.LoadScene("MainMenu");
    }
}
```

Figura 6.54 Script "ReturnToMenú"

L'script "GameMenuHandle" té funció similar a la funció "MainMenuButtonHandler" però en el menú de pausa i afegint un nou menú on s'ensenyen els controls del joc. (Veure Figura 6.55; 6.56)

```
Script de Unity | 1 referencia
public class GameMenuHandler : MonoBehaviour
{

    [SerializeField] private GameStateHandler handler;
    [SerializeField] private GameObject Menu;
    [SerializeField] private GameObject Options;
    [SerializeField] private GameObject Controls_1;
    [SerializeField] private GameObject Controls_2;
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        if (handler == null)
        {
            handler = FindObjectOfType<GameStateHandler>();
        }
    }
```

Figura 6.55 Inicialització de l'script "GameMenuHandler"

```
1 referencia
public void CloseMenu()
{
    Menu.SetActive(false);
    Options.SetActive(false);
    Controls_1.SetActive(false);
    Controls_2.SetActive(false);
}

0 referencias
public void OnReturnClick()
{
    handler.PlayerUnpauses();
}

0 referencias
public void OnControlsClick()
{
    Menu.SetActive(false);
    Controls_1.SetActive(true);
}
0 referencias
public void OnControlsClick_1()
{
    Controls_1.SetActive(false);
    Controls_2.SetActive(true);
}
0 referencias
public void OnControlsClick_2()
{
    Controls_2.SetActive(false);
    Menu.SetActive(true);
}
0 referencias
public void OnOptionsEnter()
{
    Options.SetActive(true);
    Menu.SetActive(false);
}
0 referencias
public void OnOptionsExit()
{
    Options.SetActive(false);
    Menu.SetActive(true);
}
0 referencias
public void OnExitClick()
{
    SceneManager.LoadScene("MainMenu");
}
```

Figura 6.55 Funcions de botons de l’script “GameMenuHandler”

6.10.Sistema de millora

Quan el jugador genera una estructura defensiva, aquesta ve amb una taula de millores. Aquesta permet al jugador millorar la estructura de maneres creatives. La taula està composada per una estructura central de 5 línies i columnes que s'expandeixen segons el jugador afegeixi objectes anomenats "Chips de millora" o chips. A la dreta el jugador pot veure les caracteristiques de la estructura i abaix els chips que pot utilitzar actualment. El resultat final és el que mostra la figura 6.56.

Figura 6.56 Taula de millores

6.10.1 Arrossegar objectes

El jugador ha de poder arrossegar els chips de la pantalla inicial fins als "sockets". Per afegir aquesta funcionalitat farem ús d'un sistema d'events de Unity especialitzat per aquesta funció. Per generar aquest sistema els chips tindran l'script DragDrop. La majoria d'operacions que utilitza l'script són visuals com variar el color quan el jugador està arrossegant l'objecte o mostrar una descripció quan el jugador passa el ratolí per sobre de l'objecte com mostren les Figures 6.56 i 6.57. Les funcions que s'utilitzen per altres mecàniques són les d'obtenir les dades del chip, mostrades a la figura 6.58, i les de construir o destruir el chip utilitzant recursos, mostrades a la figura 6.59.

```
Script de Unity (1 referencia de recurso) | 6 referencias
public class DragDrop : MonoBehaviour, IBeginDragHandler, IEndDragHandler, IDragHandler, IPointerEnterHandler, IPointerExitHandler
{

    [SerializeField] private ChipData data;
    [SerializeField]
    private GameObject description;
    [SerializeField] private RecipeData recipe;

    private GameObject objectDragging;
    private GameObject station;
    private ChipUIDisplay display;


    private UIRecipeHandler handler;
    private RecipeBuilder builder;

    public bool object_created = false;
    Mensaje de Unity | 0 referencias
    private void Start()
    {


        builder = GetComponent<RecipeBuilder>();
        builder.SetCurrentUIRecipe(recipe);
        station = transform.parent.parent.GetComponent<StationBoardData>().station;
        display = GetComponent<ChipUIDisplay>();
        display.SetData(data.chip_name, data.sprite);
        handler = GetComponent<UIRecipeHandler>();
        handler.DisplayRecipe(recipe);
        description.SetActive(false);
    }

    0 referencias
    public void OnDrag(PointerEventData eventData)
    {
        if (object_created)
        {
            objectDragging.transform.position = eventData.pointerCurrentRaycast.worldPosition;
        }

    }
```

Figura 6.56 Inicialització de l’script “DragDrop”

```
0 referencias
public void OnDrag(PointerEventData eventData)
{
    if (object_created)
    {
        objectDragging.transform.position = eventData.pointerCurrentRaycast.worldPosition;
    }

}
0 referencias
public void OnBeginDrag(PointerEventData eventData)
{
    if (builder.CanBuild(recipe))
    {
        object_created = true;
        objectDragging = Instantiate(data.prefab, eventData.pointerCurrentRaycast.worldPosition, this.transform.rotation, station.transform);
        if (objectDragging.GetComponent<CanvasGroup>() != null)
        {
            objectDragging.GetComponent<CanvasGroup>().alpha = 0.7f;
        }
        else
        {
            objectDragging.AddComponent<CanvasGroup>().alpha = 0.7f;
        }
    }

}

0 referencias
public void OnEndDrag(PointerEventData eventData)
{
    if (object_created)
    {
        Destroy(objectDragging);
        object_created = false;
    }


}
0 referencias
public void OnPointerEnter(PointerEventData pointerEventData)
{
    Debug.Log("Pointer");
    description.SetActive(true);
}
0 referencias
public void OnPointerExit(PointerEventData pointerEventData)
{
    description.SetActive(false);
}
```

Figura 6.57 Events de l'script "DragDrop"

```
2 referencias
public ChipData GetData()
{
    return data;
}
0 referencias
public RecipeData GetRecipe()
{
    return recipe;
}
1 referencia
public GameObject getObject()
{
    if (data != null)
    {
        return data.prefab;
    }
    return null;
}
```

Figura 6.58 Operacions "GET" de l'script "DragDrop"

```
1 referencia
public void BuildRecipe()
{
    builder.BuildRecipe(recipe);
}
1 referencia
public void DestroyRecipe()
{
    builder.DestroyRecipe(recipe);
}
```

Figura 6.59 Funcions de crear i destruir chips de l'script “DragDrop”

6.10.2 Chip

Un chip és un objecte dividit en dues parts: les seves dades i el seu efecte. Per emmagatzemar les dades de multiples chips utilitzem l'scriptable object “ChipData”, mostrat a la figura 6.60, que s'utilitzarà a l'hora de mostrar al jugador les dades del chip.

```
[CreateAssetMenu(fileName = "Chip", menuName = "Data/Chip")]
public class ChipData : ScriptableObject
{
    public int id;
    public Sprite sprite;
    public string chip_name;
    [TextArea(3, 10)]
    public string short_description;
    [TextArea(3, 10)]
    public string long_description;

    public GameObject prefab;
}
```

Figura 6.59 Scriptable object “ChipData”

A l'hora de generar multiples chips amb diferents efects, C# ens permet fer ús de poliformisme, és a dir podem crear multiples scripts diferents que hereten d'un script. (Veure Figura 6.60)

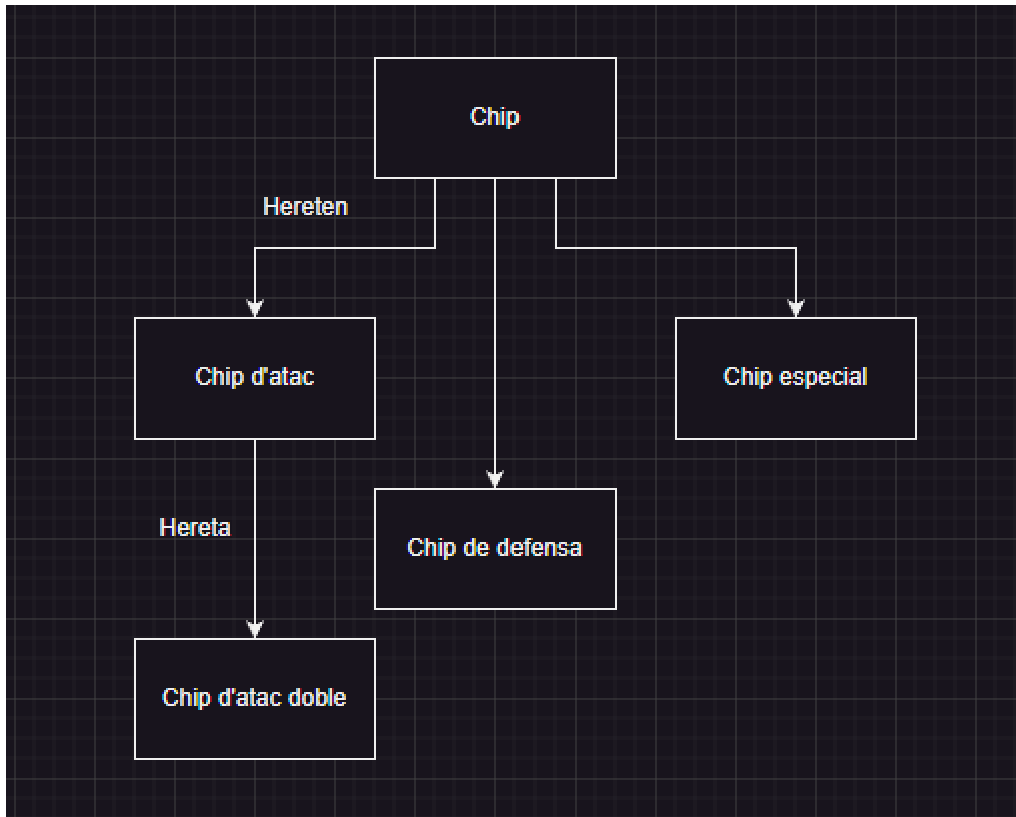

Figura 6.60 Visualització de poliformisme

L’script principal dels chips és l’script “ChipEffect”. Aquest s'encarregara de definir el tipus de chip i les variables requerides per funcionar. (Veure Figura 6.61)

```
5 referencias
public enum chip_type
{
    Damage, BuffChips, Special
}
Script de Unity | 19 referencias
public class ChipEffect : MonoBehaviour
{

    protected StateChangeHandler stat_handler;
    protected StationSocketsHandler socket_handler;
    [SerializeField] protected SocketDataHandler socket_data;


    public chip_type type;
    1 referencia
    public void SetHandlers(StateChangeHandler handler, StationSocketsHandler socket_handler,SocketDataHandler data)
    {
        this.stat_handler = handler;
        this.socket_handler = socket_handler;
        socket_data = data;
    }
    7 referencias
    public void ActivateChip()
    {

    if (stat_handler != null && socket_handler != null)
        {
            ActivateEffect();
        }

    }
```

Figura 6.61 Part “fixa” de l’script “ChipEffect”

A més, l’script té 2 funcions úniques en cada tipus de chip. L’efecte especific que té el chip al activarse i una visualització dels chips afectats per aquest.(Veure Figura 6.62)

```
4 referencias
protected virtual void ActivateEffect() { }

1 referencia
public virtual void DisplayAffected(bool type)
{
    if (GetComponent<ItemInSocket>() != null)
    {
        GetComponent<ItemInSocket>().ExternalHighlight(type);
    }

}
```

Figura 6.62 Part “variable” de l’script “ChipEffect”

6.10.2.1. Chips de millora de chips

Com aquests chips milloren altres chips, aquests utilitzen una modificació de la clase “ChipEffect” on obtenen els chips afectats. (Veure Figura 6.63)

```
4 referencias
protected virtual void GetAffectedChips()
{
    Debug.Log("Getting Chips");

    ChipEffect affected = socket_handler.GetEffect(socket_data.GetLayer(), socket_data.GetOrder() + 1);
    if (affected != null) {
        Debug.Log("An effect is being detected");
        Chips_affected.Add(affected);
    }
}
2 referencias
public void HighlightAffectedChips(bool type)
{
    for (int i=0;i<Chips_affected.Count;i++)
    {
        Chips_affected[i].DisplayAffected(type);
    }
}
```

Figura 6.63 Part “variable” de l’script “ChipModChips”

L’script base dels chips de millora és un chip que obté la informació del chip adjacent i l’activa si existeix. (Veure Figura 6.64)

```
2 referencias
protected override void ActivateEffect()
{
    Chips_affected = new List<ChipEffect>();
    GetAffectedChips();
    if (Chips_affected.Count>0)
    {
        SpecificEffect();
    }
}
2 referencias
protected virtual void SpecificEffect() {
    for (int i=0;i<Chips_affected.Count;i++)
    {
        Chips_affected[i].ActivateChip();
    }

}
```

Figura 6.64 Activació de l’script “ChipModChips”

Per aquest projecte s'han dissenyat 4 chips de millores incloent el base. A la figura 6.65 es mostra l'efecte del chip "DualDistanceMod" que activa 2 chips adjacents a la dreta del chip. A la figura 6.66 es mostra l'efecte del chip "CrossChipMod" que activa l'efecte del chip adjacent a la dreta i adalt. Finalment, a la figura 6.67 es mostra l'efecte del chip "KnightChipMod" que simula la peça d'escacs del caball i activa el chip situat dues posicions a la dreta i una abaix tres cops. Tots aquests chips hereten de la funció "ChipModChips"

```
2 referencias
protected override void GetAffectedChips()
{
    Debug.Log("Getting Chips");

    ChipEffect affected = socket_handler.GetEffect(socket_data.GetLayer(), socket_data.GetOrder() + 1);
    if (affected != null)
    {
        Debug.Log("An effect is being detected");
        Chips_affected.Add(affected);
    }
    affected = socket_handler.GetEffect(socket_data.GetLayer(), socket_data.GetOrder() + 2);
    if (affected != null)
    {
        Debug.Log("An effect is being detected");
        Chips_affected.Add(affected);
    }
}
```

Figura 6.65 Funció de l'script "DualDistanceMod"

```
2 referencias
protected override void GetAffectedChips()
{
    Debug.Log("Getting Chips");

    ChipEffect affected = socket_handler.GetEffect(socket_data.GetLayer(), socket_data.GetOrder() + 1);
    if (affected != null)
    {
        Debug.Log("An effect is being detected");
        Chips_affected.Add(affected);
    }
    if (socket_data.GetLayer() > 0)
    {
        affected = socket_handler.GetEffect(socket_data.GetLayer() - 1, socket_data.GetOrder());
        if (affected != null)
        {
            Chips_affected.Add(affected);
        }
    }
}
```

Figura 6.66 Funció de l'script "CrossChipMod"

```
2 referencias
protected override void GetAffectedChips()
{
    Debug.Log("Getting Chips");

    if (socket_data.GetLayer() < 4)
    {
        ChipEffect affected = socket_handler.GetEffect(socket_data.GetLayer() - 1, socket_data.GetOrder() +2);
        if (affected != null)
        {
            Chips_affected.Add(affected);
        }
    }
}
2 referencias
protected override void SpecificEffect()
{
    for (int i = 0; i < Chips_affected.Count; i++)
    {
        Chips_affected[i].ActivateChip();
        Chips_affected[i].ActivateChip();
        Chips_affected[i].ActivateChip();
    }

}
```

Figura 6.67 Funció de l'script "KnightChipMod"

6.10.2.2. Chips de millora de estructures

Aquests chips també utilitzen una variant de l'script "ChipEffect" ja que requereixen les dades per modificar la estructura, el tipus d'estadistica que afecta i el valor. (Veure Figura 6.68)

```
Script de Unity (2 referencias de recurso) | 1 referencia
public class ChipModTower : ChipEffect
{
    [SerializeField] protected float base_value = 2;
    [SerializeField]
    protected Stat_type stat_type;
    protected float current_value;
    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        current_value = base_value;
}
    3 referencias
    protected override void ActivateEffect()
    {
        stat_handler.modifyValue(current_value, stat_type);
    }
    0 referencias
    public void modifyValue(float amount)
    {
        current_value += amount;
        if (current_value <= 1)
        {
            current_value = 1;
        }
    }
}
```

Figura 6.68 Script "ChipModTower"

L'script base és una millora d'una característica i en aquest projecte s'ha afegit una variant que augmenta dues característiques, però al final no s'utilitza en la versió final. (Veure Figura 6.69)

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class DamageAndCritChip : ChipModTower
{

    [SerializeField]
    private Stat_type stat_type_2;
    [SerializeField] private float base_value_2 = 5;
    3 referencias
    protected override void ActivateEffect()
    {
        stat_handler.modifyValue(current_value, stat_type);
        stat_handler.modifyValue(base_value_2, stat_type_2);
    }
}
```

Figura 6.69 Script "DamageAndCritChip"

6.10.2.3. Visualització dels chips

El jugador ha de poder veure els chips en l'interficie. Per tal d'evitar haver de generar manualment cada chip. utilitzem l'script "ChipUIDisplay" mostrat en la Figura 6.70 que afegeix la imatge i nom del chip en la interfície.

```
public class ChipUIDisplay : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI  name;
    [SerializeField] private Image image;


    public void SetData(string name, Sprite image=null)
    {
        this.name.text = name;
        this.image.sprite = image;
    }
}
```

Figura 6.70 Script "ChipUIDisplay"

6.10.3. Sockets

Els sockets són els espais on el jugador arrossega els seus chips i es generen els efectes.

6.10.3.1 Estació

L'estació és l'objecte que s'encarrega de mantenir i modificar la informació dels sockets. Per poder gestionar els sockets, es guarden en la classe "SocketList" que té una llista de sockets i la funció d'afegir i eliminar sockets de la llista. (Veure Figura 6.71)

```
5 referencias
public class SocketList{
    public int layer;
    public List<SocketDataHandler> socket_data=new List<SocketDataHandler>();

    1 referencia
    public SocketList(int new_layer,SocketDataHandler handler)
    {
        layer = new_layer;
        AddSocket(handler);
    }
    2 referencias
    public void AddSocket(SocketDataHandler handler)
    {
        socket_data.Add(handler);
    }

    1 referencia
    public List<GameObject> EraseSockets(int beginning=0)
    {
        List<GameObject> objects_To_destroy=new List<GameObject>();
        int count = 0;
        for (int i=socket_data.Count-1; i>beginning ; i--)
        {
            objects_To_destroy.Add(socket_data[i].gameObject);
            count++;
            socket_data.RemoveAt(i);
        }
        socket_data[beginning].SetItem(null);
        return objects_To_destroy;
    }

}
Script de Unity (1 referencia de recurso) | 8 referencias
```

Figura 6.71 Classe"SocketList"

L’estació genera una llista de 5 “SocketList” buides com es mostra a la figura 6.72.

```
[SerializeField] private float distance_between_sockets;
[SerializeField] private GameObject socket_prefab;

[SerializeField] private Transform[] layer_origins;

private List<SocketList> Sockets=new List<SocketList>();
Mensaje de Unity | 0 referencias
private void Start()
{
    for (int i = 0; i < 5; i++)
    {
        Sockets.Add(null);
    }
}
```

Figura 6.72 Inicialització de l’script “StationSocketHandler”

Com a funcions, l’estació pot activar, eliminar i afegir sockets. Per activar sockets, busca la seva posició, obté l’script corresponent al effecte i l’activa com mostra la figura 6.73. A l’hora d’activar sockets s’activen en ordre on primer s’activen els sockets que tinguin un chip que millori altres chips, després els chips que milloren la estructura i finalment els chips especials que no estan implementats. (Veure Figura 6.74)

```
0 referencias
public void ActivateSocketEffect(int layer, int order)
{
    if(layer>=0 && order >= 0) {
        ChipEffect effect = Sockets[layer].socket_data[order].GetItem();
        if (effect != null)
        {
            effect.ActivateChip();
        }
    }

}
7 referencias
public ChipEffect GetEffect(int layer, int order)
{
    if(Sockets[layer] == null)
    {
        return null;
    }
    if (Sockets[layer].socket_data.Count-1<order)
    {
        return null;
    }
    return Sockets[layer].socket_data[order].GetItem();
}
```

Figura 6.73 Funció per activar sockets de l’script “StationSocketHandler”

```
3 referencias
public void ResetAndActivateSockets()
{
    //ResetPart
    GetComponent<StateChangeHandler>().ResetStats();

    for (int chip_types = 0; chip_types < 2; chip_types++){
        for (int i = 0; i < Sockets.Count; i++)
        {
            if (Sockets[i] != null)
            {
                for (int y = 0; y < Sockets[i].socket_data.Count-1; y++)
                {
                    Debug.Log("Loop: "+y);
                    ChipEffect current_effect = GetEffect(i, y);
                    if (current_effect != null)
                    {
                        if (chip_types == 0)
                        {
                            if (current_effect.type == chip_type.BuffChips)
                            {
                                current_effect.ActivateChip();
                            }
                        }
                        else
                        {
                            if (current_effect.type == chip_type.Damage)
                            {
                                Debug.Log(i + "  " + y);
                                current_effect.ActivateChip();
                            }
                        }
                    }
                }
            }
        }
    }

}
```

Figura 6.74 Funció per activar tots els sockets de l'script "StationSocketHandler"

Finalment, per eliminar un socket, l'estació elimina el socket de la posició elegida i tots els adjacents a la dreta i reactiva els sockets. (Veure Figura 6.75)

```
1 referencia
public void EraseSockets(int layer,int beginning = 0)
{
    List<GameObject> list =Sockets[layer].EraseSockets(beginning);
    for(int i = 0; i < list.Count; i++)
    {
        Destroy(list[i]);

    }
    list.Clear();
    ResetAndActivateSockets();
}
```

Figura 6.75 Funció per eliminar sockets de l'script "StationSocketHandler"

6.10.3.2 Socket

La funció d’un socket és poder rebre chips i activarlos si l’estació ho demana. El funcionament està dividit en dues parts: Obtenir o eliminar els chips que el jugador arrossega i guardar o esborrar la informació del chip per l'ús de l’estació. A més s’ha afegit una part d'interfície. Per la primera part utilitzem l’script “SocketDropHandler” que afegeix el chip quan el jugador l’arrossegui instanciant el chip i actualitzant les seves dades (Veure Figura 6.76) i crida a l’estació per eliminar el socket si el jugador fa clic dret. (Veure Figura 6.77)

```
Script de Unity (1 referencia de recurso) | 5 referencias
public class SocketDropHandler : MonoBehaviour, IDropHandler, IEventSystemHandler
{

    [SerializeField]
    private bool spawnsNewSocket;

    [SerializeField]
    private StateChangeHandler state_handler;

    private SocketDataHandler handler;

    [SerializeField]
    private SocketUIDisplay ui_display;

    private GameObject socket = null;
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        ui_display=GetComponent<SocketUIDisplay>();
        handler = GetComponent<SocketDataHandler>();
        state_handler = GetComponentInParent<StateChangeHandler>();
    }
    0 referencias
    public void OnDrop(PointerEventData eventData)
    {
        Debug.Log("Drop");
        if (CanAddItem() && eventData.pointerDrag.GetComponent<DragDrop>().object_created)
        {
            if (eventData.pointerDrag != null)
            {
                //Set the chip UI view
                socket = Instantiate(eventData.pointerDrag.GetComponent<DragDrop>().getObject(), transform.position, transform.rotation, transform);
                eventData.pointerDrag.GetComponent<DragDrop>().BuildRecipe();
                socket.GetComponent<CanvasGroup>().blocksRaycasts = true;
                socket.GetComponent<ItemInSocket>().InSocket = true;
                socket.GetComponent<ItemInSocket>().OriginData = eventData.pointerDrag.GetComponent<DragDrop>();
                socket.GetComponent<ChipEffect>().SetHandlers(state_handler, transform.parent.GetComponent<StationSocketsHandler>(), handler);

                //Set the chip on the data

                handler.SetItem(socket.GetComponent<ChipEffect>());

                ui_display.SetColor(1);

            }
            if (spawnsNewSocket)
            {
                StationSocketsHandler station = transform.parent.GetComponent<StationSocketsHandler>();
                station.Generate_Socket(handler.GetOrder() + 1, handler.GetLayer());
                station.ResetAndActivateSockets();
            }
        }


    }
```

Figura 6.76 Inicialització i “OnDrop” de l’script “StationSocketHandler”

```
1 referencia
public void DestroyItem() {
    if (socket != null)
    {
        socket.GetComponent<ItemInSocket>().DestroyData();
    }
}
1 referencia
public void DestroyLayer()
{
    transform.parent.GetComponent<StationSocketsHandler>().EraseSockets(handler.GetLayer(),handler.GetOrder());
    ui_display.SetColor();
}
1 referencia
public bool CanAddItem()
{
    return handler.GetItem() == null;
}
4 referencias
public void HighlightSocket(int type=1)
{
    ui_display.SetColor(type);
}
Mensaje de Unity | 0 referencias
private void OnDestroy()
{
    DestroyItem();
}
```

Figura 6.77 Funció per eliminar sockets de l'script "StationSocketHandler"

Per guardar la informació del chip, el socket utilitza l'script "SocketDataHandler", mostrat a la figura 6.78, que gestiona les dades del socket.

```
public class SocketDataHandler : MonoBehaviour
{
    [SerializeField] private int order;
    [SerializeField] private int layer;
    [SerializeField]
    private ChipEffect current_item;

    public void SetData(int new_order,int layer)
    {
        order = new_order;
        this.layer = layer;
    }

    public int GetOrder()
    {
        return order;
    }

    public int GetLayer()
    {
        return layer;
    }

    public void SetItem(ChipEffect Item)
    {
        current_item = Item;
    }

    public ChipEffect GetItem()
    {
        return current_item;
    }
}
```

Figura 6.78 Script "SocketDataHandler"

Finalment, mostrem la informació al jugador de quins sockets es veuen afectats pel chip que el jugador està senyalant amb el ratolí utilitzant l'script "SocketUIDisplay" on el socket canvia de color depenent de l'estat actual, mostrat a la figura 6.79.

```
public class SocketUIDisplay : MonoBehaviour
{
    [SerializeField]
    private Image socket_image;

    [SerializeField] private Color inactive;
    [SerializeField] private Color active;
    [SerializeField] private Color highlighted;
    [SerializeField] private Color affected;

    private void Awake()
    {
        socket_image = GetComponent<Image>();
        SetColor();
    }

    public void SetColor(int type=0)
    {
        //0 inactive
        //1 active
        //2 Highlighted

        switch (type)
        {
            case 0:
                socket_image.color = inactive;
                break;
            case 1:
                socket_image.color = active;
                break;
            case 2:
                socket_image.color = highlighted;
                break;
            case 3:
                socket_image.color = affected;
                break;
        }
    }

}
```

Figura 6.79 Script "SocketUIDisplay"

6.10.4. Estadístiques

L'estació té com a objectiu modificar les estadístiques d'una estructura. Aquestes estadístiques han de poder ser manipulades i el jugador ha de poder veure les.

6.10.4.1 Implementació

Utilitzem un scriptable object per emmagatzemar les estadistiques inicials de la estructura. (Veure Figura 6.80)

```
[CreateAssetMenu(fileName = "Tower Stats", menuName = "Data/Tower")]
public class TowerStatsData : ScriptableObject
{
    public float base_damage;
    public float base_crit_chance;
    public float base_crit_multiplayer;

    public int max_chips;

}
```

Figura 6.80 Scriptable object "TowerStatsData"

Per modificar les estadistiques base utilitzem l'script "TowerStatsHandler". Aquest s'encarrega d'inicialitzar les estadistiques base, mostrat a la figura 6.81, i manipular les, mostrat a la figura 6.82.

```
public class TowerStatsHandler : MonoBehaviour
{
    [SerializeField] private TowerStatsData base_stats;

    [SerializeField] private float damage;
    [SerializeField] private float crit_chance;
    [SerializeField] private float crit_mult;

    private int current_chips;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        ResetStats();
    }
    0 referencias
    public void SetChips(int size)
    {
        current_chips = size;
    }
```

Figura 6.81 Inicialització de l'script "TowerStatsHandler"

```
1 referencia
public void ModifyAtribute(Stat_type type, float value)
{
    switch (type)
    {
        case Stat_type.damage:
            damage=value;
            break;
        case Stat_type.crit_chance:
            crit_chance = value;

            break;
        case Stat_type.crit_mult:
            crit_mult = value;
            break;
    }
}
1 referencia
public float GetAtribute(Stat_type type)
{
    switch (type)
    {
        case Stat_type.damage:
            return damage;
        case Stat_type.crit_chance:
            return crit_chance;
        case Stat_type.crit_mult:
            return crit_mult;
    }
    return -1;
}
1 referencia
public TowerStatsData GetBaseStats()
{
    return base_stats;
}
1 referencia
public stats GetCurrentStats()
{
    stats current_stats = new stats();
    current_stats.damage = damage;
    current_stats.crit_chance = crit_chance;
    current_stats.crit_mult = crit_mult;
    return current_stats;
}
2 referencias
public void ResetStats()
{
    damage = base_stats.base_damage;
    crit_chance = base_stats.base_crit_chance;
    crit_mult = base_stats.base_crit_multiplayer;
}
```

Figura 6.82 Funcions de l’script “TowerStatsHandler”

6.10.4.2 Interficie

Per mostrar les estadístiques al jugador utilitzem l'script "UIStatText", mostrat a la figura 6.83, per estalviar manipular manualment la interfície i per afegir rapidament noves estadistiques, i l'script "UIStatHandler" que afegeix les estadistiques en una llista, mostat a la figura 6.84.

```
public class UIStatTextHandler : MonoBehaviour
{

    public Stat_type type;
    [SerializeField] private TextMeshProUGUI name;
    [SerializeField] private TextMeshProUGUI value;


    public void SetAllValues(string name, float value, Stat_type type)
    {
        SetValue(value);
        SetName(name);
        this.type = type;
    }


    public void SetValue(float value) {
        this.value.text = value.ToString();
    }

    public void SetName(string name)
    {
        this.name.text = name;
    }
}
```

Figura 6.83 Script "UIStatTextHandler"

```
public class UIStatHandler : MonoBehaviour
{
    [SerializeField] private GameObject statPrefab;

    private List<GameObject> prefabs=new List<GameObject>();


    public void SetStat(string name,float value,Stat_type type)
    {
        if (prefabs.Count == 0)
        {
            CreateStatText(name, value, type);
        }
        else
        {
            int position = FindStatPos(type);
            if (position < 0)
            {
                CreateStatText(name, value, type);
            }
            else
            {
                prefabs[position].GetComponent<UIStatTextHandler>().SetValue(value);
            }
        }
    }

    private void CreateStatText(string name, float value, Stat_type type)
    {
        GameObject stat = Instantiate(statPrefab, transform.position, transform.rotation, transform);
        stat.GetComponent<UIStatTextHandler>().SetAllValues(name, value, type);
        prefabs.Add(stat);
    }

    private int FindStatPos(Stat_type type)
    {
        for(int i = 0; i < prefabs.Count; i++)
        {
            if (prefabs[i].GetComponent<UIStatTextHandler>().type == type) return i;
        }
        return -1;
    }

    public void ResetList()
    {
        for(int i = 0; i < prefabs.Count; i++)
        {
            Destroy(prefabs[i]);
        }
        prefabs.Clear();
    }
}
```

Figura 6.84 Script “UIStatHandler”

Per poder detectar canvis en les estadistiques i actualitzar la interficie, creem l’script “StateChangeHandler” que actualitza l’script “UIStatHandler” quan les estadistiques canvien. (Veure Figura 6.85)

```
public class StateChangeHandler : MonoBehaviour
{
   [SerializeField] private TowerStatsHandler handler;
    private TowerStatsData data;

    [SerializeField] private UIStatHandler UIHandler;

    private void Awake()
    {
        data = handler.GetBaseStats();

        ResetStats();
    }

    public void modifyValue(float value,Stat_type type)
    {
        float base_value = handler.GetAtribute(type);
        float new_value = value + base_value;
        handler.ModifyAtribute(type, new_value);
        switch (type)
        {
            case Stat_type.damage:

                UIHandler.SetStat("Damage", new_value, Stat_type.damage);
                break;
            case Stat_type.crit_chance:
                UIHandler.SetStat("Critical chance", new_value, Stat_type.crit_chance);
                break;
            case Stat_type.crit_mult:
                UIHandler.SetStat("Crit multiplayer", new_value, Stat_type.crit_mult);
                break;
        }
    }

    public void ResetStats()
    {
        handler.ResetStats();

        UIHandler.ResetList();

        UIHandler.SetStat("Damage", data.base_damage,Stat_type.damage);
        UIHandler.SetStat("Critical chance", data.base_crit_chance, Stat_type.crit_chance);
        UIHandler.SetStat("Crit multiplayer", data.base_crit_multiplayer, Stat_type.crit_mult);

    }
}
```

Figura 6.85 Script “StateChangeHandler”

6.11.Jugador

En aquest projecte, el jugador controla a un personatge en primera persona amb diferents eines. En aquest apartat veurem com s'ha implementat en el projecte.

6.11.1. Estadistiques

La vida del jugador és una part essencial dels videojocs. Al arribar a 0 el jugador perd la partida. A part de la vida, el jugador té un sistema d'energia que li permet realitzar accions. Per aconseguir aquestes funcions utilitzem l'script "PlayerStateHandler". L'script inicialitza les variables del jugador, mostrat a la Figura 6.86.

```
[SerializeField] private float base_health=100f;
[SerializeField] private float base_stamina = 100f;
[SerializeField] private float base_armor = 100f;
[SerializeField]
private float stamina_recovery;
[SerializeField]
private float recovery_time = 1f;
[SerializeField]
private float run_stamina_cost;
//player current stats

private float current_health;
private float current_stamina;
private float current_armor;

//HUD
[SerializeField]
private HUDBarHandler healthSlider;
[SerializeField]
private HUDBarHandler staminaSlider;

private bool can_act=true;
private bool player_running;
private bool Is_recovering=true;

private Coroutine recoveryEnumerator;
// Start is called before the first frame update
Mensaje de Unity | 0 referencias
void Start()
{
    SetStatsToDefault();
}
```

Figura 6.86 Inicialització de l'script "PlayerStateHandler"

L'script té diverses funcions que serveixen per reduir i recuperar les estadístiques del jugador, mostrades en les Figures 6.87, 6.88 i 6.89.

```
private void Update()
{
    if (player_running)
    {
        use_stamina(run_stamina_cost * Time.deltaTime);
    }
    else
    {
        if (current_stamina < base_stamina && Is_recovering)
        {
            current_stamina += stamina_recovery * Time.deltaTime;
            if (current_stamina > base_stamina)
            {
                current_stamina = base_stamina;
            }
            if (staminaSlider != null)
            {
                staminaSlider.SetCurrentValue(current_stamina);
            }
        }

    }
}

1 referencia
private void SetStatsToDefault()
{
    current_health = base_health;
    current_stamina = base_stamina;
    current_armor =base_armor;

    if (healthSlider != null)
    {
        healthSlider.SetMaxValue(current_health);
        healthSlider.SetCurrentValue(current_health);
    }
    if (staminaSlider != null)
    {
        staminaSlider.SetMaxValue(current_health);
        staminaSlider.SetCurrentValue(current_health);
    }
}
```

Figura 6.87 Funcions de l'script "PlayerStateHandler" part 1

```
1 referencia
public void Take_Damage(float damage)
{
    float true_damage = DamageCalculator.CalculateDamage(damage, current_armor);

    Debug.Log(true_damage);
    current_health -= true_damage;
    if (healthSlider != null)
    {
        healthSlider.SetCurrentValue(current_health);
    }

    if (current_health <= 0)
    {
        FindObjectOfType<GameStateHandler>().FinishGame();
        Debug.Log("Ded");
    }
}
0 referencias
public void RecoverHealth(float amount)
{
    current_health += amount;
    if (current_health > base_health)
    {
        current_health = base_health;
    }
    if (healthSlider != null)
    {
        healthSlider.SetCurrentValue(current_health);
    }
}
2 referencias
public void use_stamina(float amount)
{
    if (CanAct())
    {
        current_stamina -= amount;

          if(recoveryEnumerator==null) recoveryEnumerator = StartCoroutine(RecoveryCooldown());
        else
        {
            StopCoroutine(recoveryEnumerator);
            recoveryEnumerator = StartCoroutine(RecoveryCooldown());
        }


        if (current_stamina<= 0)
        {
            StartCoroutine(StartRecovery());
        }
        if (staminaSlider != null)
        {
            staminaSlider.SetCurrentValue(current_stamina);
        }
    }
}
```

Figura 6.88 Funcions de l’script “PlayerStateHandler” part 2

```
0 referencias
public void recover_stamina(float amount)
{
    current_stamina += amount;
    if (staminaSlider != null)
    {
        staminaSlider.SetCurrentValue(current_stamina);
    }
}
2 referencias
public bool CanAct()
{
    return can_act;
}
2 referencias
public void SetPlayerRunning(bool running= false)
{
    player_running = running;
}
1 referencia
public bool PlayerRunning()
{
    return player_running;
}
2 referencias
public IEnumerator RecoveryCooldown()
{
    Is_recovering = false;
    yield return new WaitForSeconds(recovery_time);
    Is_recovering = true;
}
1 referencia
public IEnumerator StartRecovery()
{
    can_act = false;
    yield return new WaitUntil(() => current_stamina>=base_stamina/2);
    can_act = true;
}
1 referencia
public bool CanBeActivated(float cost)
{
    return current_stamina>=cost && can_act;
}
```

Figura 6.89 Funcions de l’script “PlayerStateHandler” part 3

6.11.2. Moviment

El jugador té 3 opcions de moviment: caminar, correr i saltar. Correr utilitza energia. Per poder moure la càmera, utilitzem l’script “cameraController”, mostrat a la figura 6.90. Aquest script rota al jugador si aquest mou el ratolí en direcció horitzontal i rota la càmera si el mou en direcció vertical.

```
Script de Unity (2 referencias de recurso) | 0 referencias
public class cameraController : MonoBehaviour
{
    [Range(0,15)] [SerializeField] float sensitivity;
    private Transform player;
    [SerializeField][Range(0,0.5f)] float smooth=0.03f;


    private Vector2 currentMouseControl=Vector2.zero;
    private Vector2 currentMouseControlVelocity = Vector2.zero;

    private float pitch=0;
    private float defaultFov=90;
    private bool stopAiming;
    public float aimSensitivity;
    private float currentSensitivity;

    //private ControlManager manager;
    Mensaje de Unity | 0 referencias
    private void Start()
    {
        //manager = FindObjectOfType<ControlManager>();
        currentSensitivity = sensitivity;

        player = transform.parent;
        //GameEvent.current.onAiming += aim;
        //GameEvent.current.notAiming += stopAim;
    }
    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void LateUpdate()
    {
        // if (!manager.inMenu())
        //  {
        if (CursorHandler.camera_enabled)
        {
            handleInput();
        }

       // }
    }
    1 referencia
    private void handleInput()
    {
        if (stopAiming)
        {
            Camera.main.fieldOfView = Mathf.Lerp(Camera.main.fieldOfView, defaultFov, 0.1f);
            if (Camera.main.fieldOfView + 1 >= defaultFov) stopAiming = false;
        }
        Vector2 targetMouseControl = new Vector2(Input.GetAxis("Mouse X"), Input.GetAxis("Mouse Y"));

        currentMouseControl = Vector2.SmoothDamp(currentMouseControl, targetMouseControl, ref currentMouseControlVelocity, smooth);
        pitch -= currentMouseControl.y * sensitivity;
        pitch = Mathf.Clamp(pitch, -90, 90);

        transform.localRotation = Quaternion.Euler(pitch, 0f, 0f);
        player.Rotate(Vector3.up * currentMouseControl.x * sensitivity);
    }
```

Figura 6.90 Script “cameraController”

A l’hora de moure al jugador utilitzem l’script “playerController”. L’script mourà al jugador en una velocitat utilitzant fisiques si camina i en una altre si corre. També informa a l’script “PlayerStateHandler” si el jugador està corrent. (Veure Figures 6.91, 6.92)

```
Mensaje de Unity | 0 referencias
private void FixedUpdate()
{
        movePlayer();
}

1 referencia
private void handleInput()
{
    Vector2 inputDirection = new Vector2(Input.GetAxisRaw("Vertical"), Input.GetAxisRaw("Horizontal")).normalized;
    walkDirection = transform.forward*inputDirection.x+transform.right*inputDirection.y;
    walkDirection.Normalize();
    if(canJump() && Input.GetKeyDown(KeyCode.Space))
    {
       body.AddForce(Vector3.up * jumpHeight, ForceMode.Impulse);
    }
    bool isMoving = walkDirection.x != 0 || walkDirection.y !=0;

    if (Input.GetKey(KeyCode.LeftShift) && isMoving && state.CanAct())
    {
        Debug.Log("Runing");
        if (state != null)
        {
            state.SetPlayerRunning(true);
        }

        isRuning = true;
    }
    else
    {
        if (state != null)
        {
            state.SetPlayerRunning(false);
        }
        isRuning = false;
    }
```

Figura 6.91 Gestió d’input de l’script “PlayerController”

```
1 referencia
private void movePlayer()
{
    float velocity;
    if (state != null) {
        if (state.PlayerRunning())
        {
            velocity = run_velocity;
        }
        else
        {
            velocity = walk_velocity;
        }
    }
    else
    {
        if (isRuning)
        {
            velocity = run_velocity;
        }
        else
        {
            velocity = walk_velocity;
        }
    }

    Vector3 dragVector =new Vector3(body.velocity.x * drag,0, body.velocity.z * drag);
    body.velocity = body.velocity + walkDirection * velocity * Time.deltaTime - dragVector;

}
```

Figura 6.92 Funció de moviment de l’script “PlayerController”

Per comprovar si el jugador pot saltar, es genera un raycast en direcció al terra i del tamany del jugador, si aquest impacta el jugador pot saltar. (Veure Figura 6.93)

```
1 referencia
public bool canJump()
{
    Debug.DrawLine(transform.position, transform.position + Vector3.down, Color.green, 0.2f);
    if (Physics.Raycast(transform.position, Vector3.down,collider.bounds.extents.y+0.1f ))
    {
        if (!collider.isTrigger)
        {
            return true;
        }

    }
    return false;
}
```

Figura 6.93 Funció de salt de l'script "PlayerController"

6.11.3. Eines

El jugador té dues eines que pot utilitzar per sobreviure: el pic i el constructor. Per poder canviar entre elles utilitzem l'script "WeaponSwitcher" que permet canviar l'eina equipada quan el jugador prem un botó. (Veure Figura 6.94)

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class WeaponSwitcher : MonoBehaviour
{

    private int current_tool;

    [SerializeField] private GameObject[] tools= new GameObject[4];

    void Start()
    {
        for(int i = 1; i < tools.Length; i++)
        {
            if (tools[i] != null)
            {
                tools[i].SetActive(false);
            }
        }
        current_tool = 0;

    }
    Mensaje de Unity | 0 referencias
    private void Update()
    {
        HandleInput();
    }
    1 referencia
    private void HandleInput()
    {
        if (Input.GetKeyDown(KeyCode.Alpha1))
        {
            ActivateTool(0);
        }
        else if (Input.GetKeyDown(KeyCode.Alpha2))
        {
            ActivateTool(1);
        }
        else if (Input.GetKeyDown(KeyCode.Alpha3))
        {
            ActivateTool(2);
        }
        else  if (Input.GetKeyDown(KeyCode.Alpha4))
        {
            ActivateTool(3);
        }
    }
    4 referencias
    private void ActivateTool(int i=0) {
        if (current_tool != i)
        {
            if (tools[i] != null)
            {
                tools[current_tool].SetActive(false);
                current_tool = i;
                tools[current_tool].SetActive(true);
            }
        }
    }
}
```

Figura 6.94 Script “WeaponSwitcher”

6.11.3.1. Pic

El pic permet al jugador atacar i destruir recursos i enemics. Està dividit en dues parts: El control i les característiques. El control gestiona l'ús de la eina i utilitza l'script "WeaponController" on el pic farà una animació i permet atacar a objectes quan el jugador prem el botó d'atac. Per poder atacar a enemics únicament quan el jugador decideix atacar, el pic té un trigger que només és actiu quan l'animació d'atac està en moviment,. (Veure Figures 6.95 i 6.96)

```
[SerializeField] private Animation animation;
[SerializeField] private BoxCollider collider;

[SerializeField] private float stamina_use = 20;
[SerializeField]
private PlayerStateHandler player_state;

private float animation_duration;
private bool can_attack = true;
private
// Start is called before the first frame update
Mensaje de Unity | 0 referencias
void Start()
{
    if (collider == null)
    {
        collider = GetComponent<BoxCollider>();
    }
    if (animation == null)
    {
        animation = GetComponent<Animation>();
    }
    if (player_state == null)
    {
        player_state = FindObjectOfType<PlayerStateHandler>();
    }
    animation_duration = animation.clip.length;
    collider.enabled = false;
}

// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    HandleInput();
}
```

Figura 6.95 Inicialització de l'script "WeaponController"

```
// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    HandleInput();
}
1 referencia
private void HandleInput()
{
    if (Input.GetMouseButtonDown(0) && player_state.CanBeActivated(stamina_use))
    {
        if (can_attack)
        {
            if (player_state != null)
            {
                player_state.use_stamina(stamina_use);
            }

            collider.enabled = true;
            animation.Play();
            StartCoroutine(Attack());
        }

    }
}
1 referencia
IEnumerator Attack()
{
    can_attack = false;
    yield return new WaitForSeconds(animation_duration);
    collider.enabled = false;
    can_attack = true;
}

Mensaje de Unity | 0 referencias
private void OnDisable()
{

    animation.Stop();
    can_attack = true;
    collider.enabled = false;
    transform.position = transform.parent.position;
    transform.rotation = transform.parent.rotation;


}
0 referencias
```

Figura 6.96 Funcions de l’script “WeaponController”

A l’hora d’atacar el pic ha de detectar si un objecte ha entrat en el seu trigger i si és un objecte que pot rebre dany, aplicar aquest dany. Utilitzem l’script “WeaponBehaviour”, mostrat a la Figura 6.97.

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class WeaponBehaviour : MonoBehaviour
{

    [SerializeField] private float enemy_dmg=10;
    [SerializeField] private float resource_dmg=10;

    [SerializeField] private BoxCollider collider;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        if (collider == null)
        {
            collider = GetComponent<BoxCollider>();
        }
    }
    Mensaje de Unity | 0 referencias
    private void OnTriggerEnter(Collider other)
    {
        if (other.tag == "Enemy")
        {
            if (other.gameObject.GetComponent<EnemyHandler>())
            {
                other.gameObject.GetComponent<EnemyHandler>().Take_Damage(enemy_dmg);
                collider.enabled = false;
            }

        }
        else if (other.tag == "Resource")
        {
            if (other.gameObject.GetComponent<BreakObject>())
            {
                other.gameObject.GetComponent<BreakObject>().Take_Damage(resource_dmg);
                collider.enabled = false;
            }

        }
    }
}
```

Figura 6.97 Script “WeaponBehaviour”

6.11.3.2. Constructor

El constructor és una eina que utilitza recursos per construir estructures defensives. Volem permetre poliformisme a l'hora de crear eines, Per tant utilitzeml'script "BasicTool" que serveix com a base per a eines. (Veure Figura 6.97)

```
public class BasicTool : MonoBehaviour
{
    [SerializeField] protected float distance;
    [SerializeField] private bool is_active = false;

    // Start is called before the first frame update

    void Start()
    {

    }

    // Update is called once per frame

    void Update()
    {
        if (is_active)
        {
            HandleEffect();
        }

    }

    protected virtual void HandleEffect() { }

    public void Activate()
    {
        is_active = true;
    }

    public void Deactivate()
    {
        is_active = false;
    }
}
```

Figura 6.97 Script "BasicTool"

El constructor utilitza un raycast per apuntar a la zona on crear la estructura defensiva. El raycast només detecta la máscara del terra per evitar construir en zones no desitjades. Si el jugador té suficients materials per a generar la estructura i prem la tecla "R", es construirà la estructura consumint els materials requerits. (Veure Figura 6.98)

```
2 referencias
protected override void HandleEffect() {
    Aim();
}
1 referencia
private void Aim()
{
    RaycastHit hit;
    if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit, distance, BuildMask))
    {
        Debug.DrawRay(transform.position, transform.TransformDirection(Vector3.forward) * hit.distance, Color.blue);
        if (Input.GetKeyDown(KeyCode.R))
        {
            //Destroy(hit.collider.gameObject);
        }
    }
    // Does the ray intersect any objects excluding the player layer
    else if (Physics.Raycast(transform.position, transform.TransformDirection(Vector3.forward), out hit, distance, GroundMask))
    {
        Debug.DrawRay(transform.position, transform.TransformDirection(Vector3.forward) * hit.distance, Color.blue);
        if (Input.GetKeyDown(KeyCode.R))
        {
            if (handler.CanBuild(current_recipe))
            {
                Instantiate(current_recipe.item, hit.point + Vector3.up, Quaternion.identity);
                if (handler != null)
                {
                    handler.BuildRecipe(current_recipe);
                }
            }
            else
            {
                Debug.Log("Not enough resources");
            }


        }
    }

    else
    {
        Debug.DrawRay(transform.position, transform.TransformDirection(Vector3.forward) * distance, Color.red);
        if (Input.GetKeyDown(KeyCode.R))
        {
            //Add on the maximum distance
        }
    }

}
```

Figura 6.98 Funció Aim de “BuilderController”

6.12. Gestió de recursos

Els recursos són objectes que el jugador pot obtenir destruint objectes de l'escenari i enemics. Aquests es poden utilitzar per construir estructures defensives o chips. Cada objecte que es pot construir amb recursos ha de tenir una recepta amb la llista de recursos que requereix.

6.12.1 Recursos

Per guardar les dades utilitzem l'scriptable object "ResourceData" on es guarda el tipus de recurs, el nom, la imatge i el nombre màxim de recursos, mostrat a la Figura 6.99.

```
[CreateAssetMenu(fileName = "Resource_Data", menuName = "Data/Resource", order = 1)]
public class ResourceData : ScriptableObject
{
    public enum ResourceType
    {
        Upgrade,currency
    }

    public string ID;
    public ResourceType type;

    public Sprite sprite;
    public string name;
    public int max_stacks;

}
```

Figura 6.99 ScriptableObject "ResoureData"

Per utilitzar i gestionar recursos utilitzem la classe "Resource". Aquesta classe té totes les funcionalitats requerides per utilitzar recursos: Afegir i reduir el número de recursos, obtenir el número actual i el número que l'identifica. (Veure 6.100)

```
5 referencias
public class Resource
{

    [SerializeField] private ResourceData data;
    [SerializeField] private int amount;


    1 referencia
    public Resource (ResourceData resource)
    {
        data = resource;
        amount = 0;
    }

    2 referencias
    public int IncreaseAmount(int amount)
    {
        //Returns the amount that doesnt fit

        this.amount += amount;
        if (this.amount > data.max_stacks)
        {
            int excess = this.amount - data.max_stacks;
            this.amount = data.max_stacks;
            return excess;
        }
        return 0;
    }
    1 referencia
    public int DecreaseAmount(int amount)
    {
        //Returns the amount
        this.amount -= amount;
        return this.amount;
    }
    0 referencias
    public bool IsEmpty()
    {
        return amount == 0;
    }
    1 referencia
    public string GetID()
    {
        return data.ID;
    }
    1 referencia
    public int GetAmount()
    {
        return amount;
    }
}
```

Figura 6.100 Classe “Resource”

Els recursos s'obtindran quan un enemic o objecte destructible és destruït. Per aconseguir aquesta funcionalitat utilitzem l'script "ResourceDropManager" que s'afegeix a tots els objectes que donen recursos, mostrat a la Figura 6.101. Aquest script rep un recurs i afegeix un número aleatori entre dos valors escollits al inventari.

```
Script de Unity (6 referencias de recurso) | 2 referencias
public class ResourceDropManager : MonoBehaviour
{
    [SerializeField] private ResourceData resourceData;
    [SerializeField] private int min_drop;
    [SerializeField] private int max_drop;


    private ResourceManager manager;

    Mensaje de Unity | 0 referencias
    private void Awake()
    {
        manager = FindObjectOfType<ResourceManager>();
    }
    2 referencias
    public void SpawnResource()
    {
        manager.addItem(resourceData, Random.Range(min_drop, max_drop));
        GameEvents.current.ResourceModified(resourceData);

    }


    Mensaje de Unity | 0 referencias
    private void OnValidate()
    {
        if (min_drop < 0)
        {
            min_drop = 0;
        }
        if (min_drop > max_drop)
        {
            min_drop = max_drop;
        }
        if (max_drop < 0)
        {
            max_drop = 0;
        }
        if (max_drop < min_drop)
        {
            max_drop = min_drop;
        }
    }
}
```

Figura 6.101 Script "ResourceDropManager"

6.12.2 Inventari

Per gestionar els recursos de la partida utilitzem l'script "ResourceManager, mostrat a les figures 6.102 i 6.103. Aquest script té funcions per afegir, eliminar, modificar i obtenir dades de recursos.

```
Script de Unity (1 referencia de recurso) | 8 referencias
public class ResourceManager : MonoBehaviour
{
    [SerializeField] private List<Resource> resources=new List<Resource>();

    2 referencias
    public void addItem(ResourceData data, int amount=1)
    {
        Debug.Log("Adding: " + data.name);
        if (amount <= 0)
        {
            return;
        }
        int position = FindPositionInInventory(data);
        int excess = amount;
        if (position >= 0)
        {
            excess = resources[position].IncreaseAmount(excess);
        }
        else
        {
            Resource resource= new Resource (data);
            excess=resource.IncreaseAmount(excess);
            resources.Add(resource);
            Debug.Log("Added item on position: " + (resources.Count-1));
        }
        if (excess > 0)
        {
            Debug.Log("There is an excess of: " + excess);
        }
    }
```

Figura 6.102 Script "ResourceManager" part 1

```
1 referencia
public void RemoveItem(ResourceData data, int amount = 1)
{
    if (amount <= 0)
    {
        return;
    }
    int position = FindPositionInInventory(data);
    int resourceLeft;
    if (position >= 0)
    {
        resourceLeft = resources[position].DecreaseAmount(amount);
        Debug.Log("Items remaining " + resourceLeft);
        if (resourceLeft == 0)
        {
            resources.RemoveAt(position);
        }
        else if (resourceLeft < 0)
        {
            Debug.LogError("Something is wrong with the removal of items");
        }
    }
    else
    {
        Debug.LogError("Item Doesnt exist");
    }
}
3 referencias
public int FindPositionInInventory(ResourceData data)
{
    string ID = data.ID;
    for(int i=0;i< resources.Count;i++)
    {
        if (resources[i].GetID() == ID)
        {
            return i;
        }
    }
    return -1;
}

4 referencias
public int ResourceAmount(ResourceData data)
{
    int pos = FindPositionInInventory(data);
    if (pos == -1) return 0;
    return resources[pos].GetAmount();
}
```

Figura 6.103 Script “ResourceManager” part 2

6.12.3 Receptes

Les receptes s'utilitzen a l'hora de crear estructures o chips. Les dades es guarden en l'scriptable object “RecipeData”, mostrat a la figura 6.104. I per permetre al jugador veure les receptes utilitzem l'script “UIRecipe”, mostrat a la figura 6.105 i l'script “UIRecipeHandler”, mostrat a la figura 6.106.

```
[System.Serializable]
public class RecipeResource
{
    public ResourceData resource;
    public int amount;
}


[CreateAssetMenu(fileName = "Recipe_data", menuName = "Data/Recipe", order = 1)]
public class RecipeData : ScriptableObject
{
    public RecipeResource[] resources;
    public GameObject item;
}
```

Figura 6.104 ScriptableObject “RecipeData”

```
public class UIRecipe : MonoBehaviour
{
    [SerializeField] private TextMeshProUGUI name;
    [SerializeField] private TextMeshProUGUI required;
    [SerializeField] private TextMeshProUGUI obtained;
    [SerializeField] private Image sprite;



    public void GenerateRecipe(RecipeResource data, int obtained)
    {
        name.text = data.resource.name;
        required.text = data.amount.ToString();
        this.obtained.text = obtained.ToString();
        sprite.sprite = data.resource.sprite;
    }

    public void UpdateObtained(int obtained)
    {
        this.obtained.text = obtained.ToString();
    }
}
```

Figura 6.105 Script “UIRecipe”

```
public class UIRecipeHandler : MonoBehaviour
{
    [SerializeField] private GameObject display;
    [SerializeField]
    private ResourceManager manager;
    private List<GameObject> display_list=new List<GameObject>();
    [SerializeField] private GameObject Grid;


    private void Awake()
    {
        manager = FindObjectOfType<ResourceManager>();
    }


    public void DisplayRecipe(RecipeData data)
    {
        if (display_list.Count > 0)
        {
            CleanDisplay();
        }
        for(int i = 0; i < data.resources.Length; i++)
        {
            GameObject newDisplay=Instantiate(display, Grid.transform);
            UIRecipe recipe = newDisplay.GetComponent<UIRecipe>();
            if (recipe != null && manager != null)
            {

                recipe.GenerateRecipe(data.resources[i],manager.ResourceAmount(data.resources[i].resource));
            }
            display_list.Add(newDisplay);
        }
    }

    private void CleanDisplay()
    {
        for (int i = 0; i < display_list.Count; i++)
        {
            Destroy(display_list[i]);
        }
        display_list.Clear();
    }

    public void ModifyDisplay(int position,int amount)
    {
        UIRecipe recipe = display_list[position].GetComponent<UIRecipe>();
        if (recipe != null && manager != null)
        {

            recipe.UpdateObtained(amount);
        }
    }
}
```

Figura 6.106 Script “UIRecipeHandler”

Pel funcionament de les receptes utilitzem l’script “RecipeBuilder”. Aquest script té la funció de construir i destruir objectes i de mostrar la recepta de l’estructura al jugador si s'està construint. (Veure Figures 6.107; 6.108; 6.109)

```
[SerializeField]
private ResourceManager manager;

[SerializeField]
private UIRecipeHandler UImanager;

private RecipeData currentData;

Mensaje de Unity | 0 referencias
private void Awake()
{
    if (manager == null)
    {
        manager = FindObjectOfType<ResourceManager>();
    }
    if (UImanager == null)
    {
        UImanager = FindObjectOfType<UIRecipeHandler>();
    }
}
Mensaje de Unity | 0 referencias
private void Start()
{
   GameEvents.current.onResourceModified += ModifyDisplay;
}
3 referencias
public void SetCurrentUIRecipe(RecipeData data)
{
    currentData = data;
    if (UImanager!=null)
    {

        UImanager.DisplayRecipe(currentData);
    }
}
```

Figura 6.107 Inicialització de l'script "RecipeBuilder"

```
3 referencias
public void SetCurrentUIRecipe(RecipeData data)
{
    currentData = data;
    if (UImanager!=null)
    {

        UImanager.DisplayRecipe(currentData);
    }
}

2 referencias
public bool CanBuild(RecipeData data)
{
    for(int i=0; i < data.resources.Length; i++) {
        if (data.resources[0].amount>manager.ResourceAmount(data.resources[0].resource)) return false ;
    }
    return true;
}
2 referencias
public void BuildRecipe(RecipeData data)
{
    for (int i = 0; i < data.resources.Length; i++)
    {
        manager.RemoveItem(data.resources[i].resource,data.resources[i].amount);
        GameEvents.current.ResourceModified(data.resources[i].resource);
    }
}
1 referencia
public void DestroyRecipe(RecipeData data)
{
    for (int i = 0; i < data.resources.Length; i++)
    {
        manager.addItem(data.resources[i].resource, data.resources[i].amount);
        GameEvents.current.ResourceModified(data.resources[i].resource);
    }
}
```

Figura 6.108 Funcions de gestió de receptes de l’script “RecipeBuilder”

```
public void ModifyDisplay(ResourceData data) {
    if (UImanager != null)
    {
        for(int i = 0; i < currentData.resources.Length; i++)
        {
            if (currentData.resources[i].resource.ID == data.ID)
            {
                UImanager.ModifyDisplay(i, manager.ResourceAmount(data));
                return;
            }
        }
    }
}
Mensaje de Unity | 0 referencias
public void OnEnable()
{
    //UImanager.gameObject.SetActive(true);
    if (currentData != null)
    {
        SetCurrentUIRecipe(currentData);
    }
    if(UImanager.gameObject.GetComponent<Animator>() != null)
    {
        UImanager.gameObject.GetComponent<Animator>().Play("RecipeDisplayActivate");
    }
}
Mensaje de Unity | 0 referencias
public void OnDisable()
{
    //UImanager.gameObject.SetActive(false);
    if (UImanager.gameObject.GetComponent<Animator>() != null)
    {
        UImanager.gameObject.GetComponent<Animator>().Play("RecipeDisplayDeactivate");
    }
}
```

Figura 6.109 Part d’interficie de l’script “RecipeBuilder”

6.13. Estructures defensives

Per defensar se dels enemics, el jugador pot crear estructures defensives. Per aquest projecte s’ha dissenyat la torre, que pot escollir objectius en una area i atacar.

6.13.1. Escollir objectiu

Per escollir l'enemic farem ús d'un "sphere cast". El comportament de les torres s'ha inspirat en el disseny d'estructures del videojoc Bloons TD 6 on les torres poden elegir com a objectiu el primer, l'últim, el més fort i el més proper a la torre. En el nostre projecte només pot escollir com a objectiu el més proper, però l'script permet escollir entre les 4 opcions, en cas d'un videojoc final. (Veure Figures 6.110; 6.111; 6.112)

```
5 referencias
private enum TargettingMode
{
    First, Last, MostHealth, Closest
}

[SerializeField] private float radius;
[SerializeField] private LayerMask detectionLayer;
[SerializeField] private Transform sphereCenter;


[SerializeField] private TargettingMode currentMode;
[SerializeField] private TowerAttackBase turret;
// Start is called before the first frame update
Mensaje de Unity | 0 referencias
void Start()
{
    if (sphereCenter == null)
    {
        sphereCenter = transform;
    }
}
```

Figura 6.110 Inicialització de l'script "EnemyDetection"

```
// Update is called once per frame
Mensaje de Unity | 0 referencias
void Update()
{
    Collider[] hitColliders = Physics.OverlapSphere(sphereCenter.position, radius, detectionLayer);

    if (hitColliders.Length > 0)
    {
        chooseTarget(hitColliders);
    }

}
0 referencias
public float getRadius()
{
    return radius;
}
Mensaje de Unity | 0 referencias
private void OnDrawGizmos()
{
    Gizmos.DrawWireSphere(sphereCenter.position, radius);
}
```

Figura 6.111 Spherecast de l'script "EnemyDetection"

```
1 referencia
private void chooseTarget(Collider[] hitColliders)
{
    int lastPosition = hitColliders.Length - 1;
    switch (currentMode)
    {
        case TargettingMode.First:
            handleDetection(hitColliders[0]);
            break;
        case TargettingMode.Last:
            handleDetection(hitColliders[lastPosition]);
            break;
        case TargettingMode.MostHealth:
            handleDetection(hitColliders[0]);
            break;
        case TargettingMode.Closest:
            int targetPosition=0;
            float closest= Vector3.Distance(sphereCenter.position, hitColliders[targetPosition].transform.position);
            for (int i = 1; i < hitColliders.Length; i++)
            {
                float newDistance= Vector3.Distance(sphereCenter.position, hitColliders[i].transform.position);
                if (newDistance < closest)
                {
                    closest = newDistance;
                    targetPosition = i;
                }
            }
            handleDetection(hitColliders[targetPosition]);
            break;
    }
}
4 referencias
private void handleDetection(Collider entity)
{
    turret.AttackTarget(entity.gameObject.transform);
    Debug.DrawRay(sphereCenter.position, entity.transform.position, Color.red);
}
```

Figura 6.112 Escollir objectiu de l’script “EnemyDetection”

6.13.2. Atacar

Un cop escollit l’objectiu, la torre ataca, instanciant projectils amb les estadístiques definides en la estació de modificació i l’indica com a objectiu l’enemic. (Veure Figura 6.113)

```
public class TowerAttackBase : MonoBehaviour
{
    [SerializeField] private float TimeBetweenShots;
    [SerializeField] private GameObject prefab;
    [SerializeField] private Transform spawner;

    [SerializeField] private TowerStatsHandler stats_handler;

    private bool canShoot=true;
    // Start is called before the first frame update

    void Start()
    {
        if (spawner == null)
        {
            spawner = transform;
        }
    }

    public void AttackTarget(Transform target)
    {
        if (!canShoot)
        {
            return;
        }
        Attack(target);
        StartCoroutine(StartCooldown());
    }

    public void Attack(Transform target)
    {
        BaseDamageProjectile projectile_stats= new BaseDamageProjectile(); //For now
        stats data = stats_handler.GetCurrentStats();
        projectile_stats.base_damage = data.damage;
        projectile_stats.crit_chance = data.crit_chance;
        projectile_stats.crit_multiplayer = data.crit_mult;
        GameObject newBullet = Instantiate(prefab, spawner.position, Quaternion.identity);
        newBullet.GetComponent<BasicProjectileBehaviour>().AddEffect(projectile_stats);
        newBullet.GetComponent<BasicProjectileBehaviour>().setTarget(target);

}

    public IEnumerator StartCooldown()
    {
        canShoot = false;
        yield return new WaitForSeconds(TimeBetweenShots);
        canShoot = true;
    }
}
```

Figura 6.113 Script “TowerAttackBase”

Aquest script pot ser utilitzat per crear altres torres amb diferents efectes. Només caldria un script que hereti l’script “TowerAttackBase” i modificar la funció “Attack”.

6.13.3. Projectil

Un projectil és un objecte que es mou i genera un efecte al impactar amb un objectiu. És convenient que en un joc amb torres defensives, aquest projectils segueixin a l'enemic.

6.13.3.1. Comportament

Pel comportament del projectil utilitzarem l'script “BasicProjectileBehaviour”. El projectil es mou cap a un objectiu definit quan es crea i al impactar li aplica una llista d’efectes. Per moure’s cap a l’objectiu, el projectil rota periòdicament en la direcció del objectiu i es mou endavant. Si el projectil no ha impactat en un temps determinat, aquest es destrueix. (Veure Figures 6.114; 6.115)

```
[SerializeField] private float speed;
[SerializeField] private float lifetime;

private Transform target;
private Rigidbody body;

private Vector3 direction;
private string EffectTag;
private List<ProjectileEffectBase> effects = new List<ProjectileEffectBase>();
Mensaje de Unity | 0 referencias
private void Start()
{
    body = GetComponent<Rigidbody>();
    StartCoroutine(TimerRoutine());
}
// Update is called once per frame
Mensaje de Unity | 0 referencias
private void FixedUpdate()
{
    aimAtTarget();
    moveProjectile();
}
```

Figura 6.114 Inicialització de l’script “TowerAttackBase”

```
Mensaje de Unity | 0 referencias
private void OnTriggerEnter(Collider other)
{
    if (other.tag == EffectTag && other.GetComponent<EnemyHandler>())
    {
        OnHit(other.gameObject);
    }
}
1 referencia
private void OnHit(GameObject hit)
{
    for(int i = 0; i < effects.Count; i++)
    {
        effects[i].OnHit(hit);
    }
    Destroy(this.gameObject);
}
1 referencia
private void aimAtTarget()
{
    if (target != null)
    {
        transform.LookAt(target.transform);
    }
    direction = transform.forward;
}
1 referencia
private void moveProjectile()
{
    body.velocity = direction * speed * Time.fixedDeltaTime;
}
1 referencia
public void setTarget(Transform target)
{
    this.target = target;
    EffectTag = target.tag;
}
1 referencia
public void AddEffect(ProjectileEffectBase effect)
{
    effects.Add(effect);
}
1 referencia
private IEnumerator TimerRoutine()
{
    //code can be executed anywhere here before this next statement
    yield return new WaitForSeconds(lifetime); //code pauses for 5 seconds
    Destroy(this.gameObject);

}
```

Figura 6.115 Funcions de l’script “TowerAttackBase”

6.13.3.2. Efectes

A L'inici del projecte, els projectils tindrien diferents efectes als enemics: congelar, verí, etc. Però aquesta idea es va descartar en el desenvolupament amb l'objectiu de enfocarse en parts més importants pel disseny global. I per tant l'únic efecte és el de fer mal fixe a l'enemic. L'efecte de danyar a l'enemic és l'script "BaseDamageProjectile" que hereta de la clase "ProjectilesEffectBase", mostrat a la Figura 6.116, que genera un atac al enemic segons les estadístiques del projectil. (Veure Figura 6.117)

```
Script de Unity | 4 referencias
public abstract class ProjectileEffectBase : MonoBehaviour
{
    public float base_damage;
    public float crit_chance;
    public float crit_multiplayer;

    1 referencia
    public void OnHit(GameObject hit)
    {
        applyEffectToTarget(hit);
    }
    2 referencias
    protected virtual void applyEffectToTarget(GameObject target)
    {

    }
    1 referencia
    protected float calulateDamage()
    {
        //Math is a bit shit lolololol
        bool is_crit = Random.Range(0, 1) <= crit_chance;
        if (is_crit) return base_damage * crit_multiplayer;
        else return base_damage;
    }
}
```

Figura 6.116 Classe "ProjectileEffectsBase"

```
public class BaseDamageProjectile : ProjectileEffectBase
{

    protected override void applyEffectToTarget(GameObject target)
    {
        target.GetComponent<EnemyHandler>().Take_Damage(calulateDamage());
    }
}
```

Figura 6.117 Script "BaseDamageProjectile"

6.14. Generació procedural

La generació procedural es refereix a generar, en aquest cas, un mapa aleatòriament utilitzant una sèrie d'instruccions o regles. En videojocs solen haver dos tipus diferents: Generació d'habitacions procedurals i generació de terreny procedural. La generació d'habitacions procedurals es basa en crear diverses habitacions a mà i unir les amb diverses regles. La generació de terreny procedural es basa en utilitzar "Noise", una textura amb valors entre 0 i 1 aleatoris, i regles per generar un terreny. El projecte utilitza la generació de terreny procedural.

6.14.1 Noise

"Noise" és una textura generada aleatoriament a partir d'un número que anomenarem llavor. (Veure Figura 6.118)

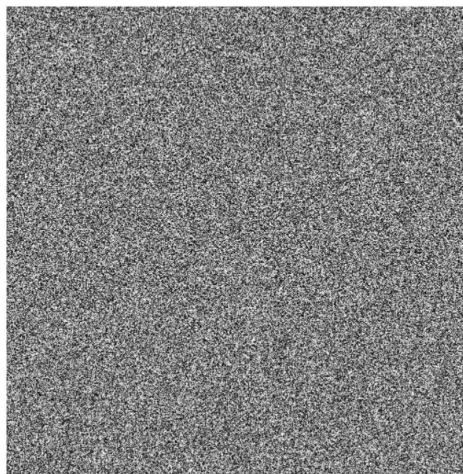

Figura 6.118 Visualització de noise

Ja que un "Noise" o soroll és molt aleatori, no és molt útil per generar terreny. Per aquesta raó s'utilitzen diferents variants de noise. Les variants més conegudes s'anomenen "Perlin noise" i "Voronoi noise". (Veure Figures 6.119; 6.120)

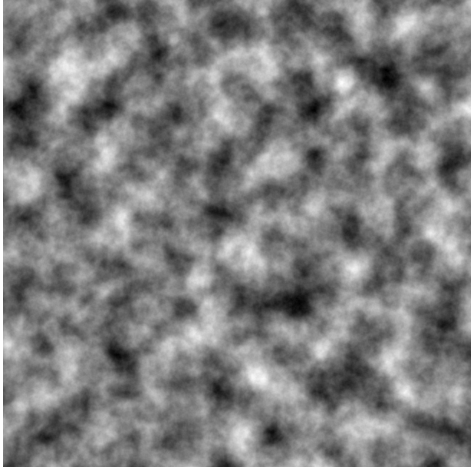

Figura 6.119 Perlin Noise

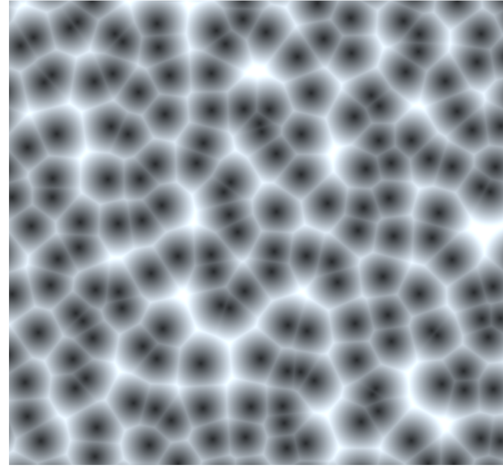

Figura 6.120 Voronoi Noise

Degut a les propietats del perlin noise, utilitzem aquest en el projecte. Per tal de generar una textura més semblant a un terreny, utilitzarem propietats anomenades “Octaves”, “Persistance” i “Lacunarity”. L’efecte que genera és el mostrat a la figura 6.121

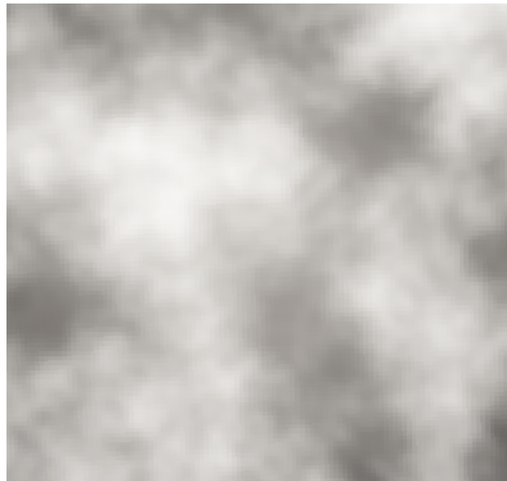

Figura 6.121 Perlin Noise alterat

Finalment, per generar l’efecte d’una illa, crearem un efecte de “Falloff” en la textura,. L’efecte que genera és el mostrat a la figura 6.122.

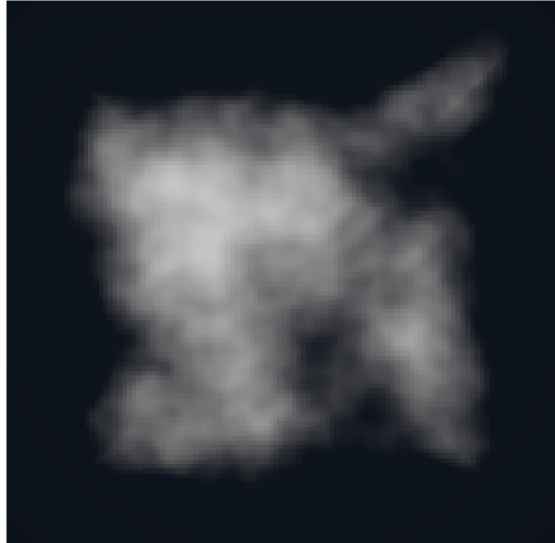

Figura 6.122 Textura final

La funció que genera aquesta textura és el mostrat a les figures 6.123 i 6.124.

```
1 referencia
public static float[,] GenerateNoiseMap(int map_width, int map_height, int seed ,float noise_scale, int octaves, float persistance, float lacunarity, Vector2 offset)
{
    float[,] noise_map = new float[map_width, map_height];

    System.Random prng = new System.Random(seed);
    Vector2[] octave_offsets = new Vector2[octaves];
    for(int i = 0; i < octaves; i++)
    {
        float offset_X = prng.Next(-100000, 100000)+ offset.x;
        float offset_Y = prng.Next(-100000, 100000) + offset.y;
        octave_offsets[i] = new Vector2(offset_X, offset_Y);
    }

    if (noise_scale <= 0)
    {
        noise_scale = 0.0001f;
    }

    float max_noise_height = float.MinValue;
    float min_noise_height = float.MaxValue;

    float half_width = map_width / 2f;
    float half_height = map_height / 2f;

    for (int x=0; x < map_width; x++)
    {
        for (int y = 0; y < map_height; y++)
        {
            float amplitude = 1;
            float frequency = 1;
            float noise_height = 0;

            for (int i= 0;i< octaves; i++)
            {
                float sample_x = (x-half_width) / noise_scale*frequency + octave_offsets[i].x;
                float sample_y = (y - half_width) / noise_scale*frequency + octave_offsets[i].y;

                float perlin_value = Mathf.PerlinNoise(sample_x, sample_y) * 2 - 1;
                noise_height += perlin_value * amplitude;

                amplitude *= persistance;
                frequency *= lacunarity;

            }
            if (noise_height > max_noise_height)
            {
                max_noise_height = noise_height;
            }
            else if (noise_height < min_noise_height)
            {
                min_noise_height = noise_height;
            }
```

Figura 6.123 Creació de “Noise” part 1

```
            }
            if (noise_height > max_noise_height)
            {
                max_noise_height = noise_height;
            }
            else if (noise_height < min_noise_height)
            {
                min_noise_height = noise_height;
            }
            noise_map[x, y] = noise_height;

        }
    }
    for (int x = 0; x < map_width; x++)
    {
        for (int y = 0; y < map_height; y++)
        {
            noise_map[x, y] = Mathf.InverseLerp(min_noise_height, max_noise_height, noise_map[x, y]);
        }
    }
            return noise_map;
}
```

Figura 6.124 Creació de “Noise” part 2

Per generar la llavor s’agafa un número aleatori al inici del joc, mostrat a la Figura 6.125.

```
public static class GameSeed
{
    public static int currentSeed=-1;

    public static void GenerateSeed()
    {
        currentSeed = Random.Range(0, 9999999);
    }

    public static void SetSeed(int seed)
    {
        currentSeed = seed;
    }
}
```

Figura 6.125 Generació de llavor

Finalment, per generar l’efecte de falloff s’utilitza l’script “FallOffGenerator”, mostrat a la figura 6.126.

```
public static class FalloffGenerator
{
    public static float[,] GenerateFalloffMap(int size)
    {
        float[,] map = new float[size, size];

        for(int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                float x = i / (float)size*2-1;
                float y = j / (float)size * 2 - 1;

                float value = Mathf.Max(Mathf.Abs(x), Mathf.Abs(y));

                map[i, j] = Evaluate(value);
            }
        }
        return map;
    }


    static float Evaluate(float value)
    {
        float a = 3;
        float b = 2.2f;

        return Mathf.Pow(value, a) / (Mathf.Pow(value, a) + Mathf.Pow(b - b * value, a));
    }
}
```

Figura 6.126 Generació de “Falloff”

6.14.2 Mesh

Un mesh és un objecte format per la unió de triangles en successió. Cada punt del triangle té una posició en l’espai. Considerem el cas que mostra la figura 6.127. En aquest cas tindriem un mesh format pels triangles 1,3,2 i 1,4,2.

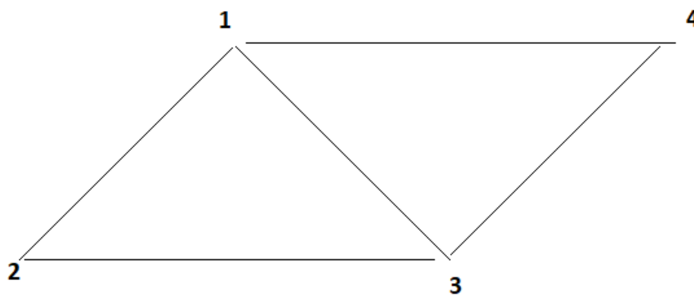

Figura 6.127 Visualització d’un mesh

Utilitzant aquesta tècnica, podem generar una llista de posicions en l'espai i unirles per triangles formant un terreny. Per tal de generar variació en alçada, podem considerar el noise que hem generat anteriorment. Cada pixel d'aquell noise representa un punt en l'espai, i el valor del pixel multiplicat per un valor representa l'alçada. Combinant els punts en l'espai que genera el noise i unint los en un mesh, podem generar un terreny. (Veure Figura 6.128 i 6.129)

```
5 referencias
public class MeshData
{
    public Vector3[] vertices;
    public Vector2[] UVs;
    public int[] triangles;

    private int triangle_index;
    1 referencia
    public MeshData(int width,int height){
        vertices = new Vector3[width * height];
        UVs = new Vector2[width * height];
        triangles = new int[(width - 1) * (height - 1)*6];
    }
    2 referencias
    public void AddTriangle(int a,int b, int c)
    {
        triangles[triangle_index] = c;
        triangles[triangle_index+1] = b;
        triangles[triangle_index+2] = a;
        triangle_index += 3;
    }
    1 referencia
    public Mesh CreateMesh()
    {
        Mesh mesh = new Mesh();
        mesh.vertices = vertices;
        mesh.triangles = triangles;
        mesh.uv = UVs;
        mesh.RecalculateNormals();
        return mesh;
    }
}
```

Figura 6.128 Classe "MeshData"

```
1 referencia
public static class MeshGenerator
{
    1 referencia
    public static MeshData GenerateTerrainMesh(float [,] heightMap, float height_multiplier, AnimationCurve height_curve,int LOD)
    {
        int width = heightMap.GetLength(0);
        int height = heightMap.GetLength(1);

        float top_left_X = (width - 1) / -2f;
        float top_left_Z = (height - 1) / 2f;

        MeshData data = new MeshData(width, height);
        int vertex_index = 0;

        int mesh_simplification_increment =(LOD==0)?1 : LOD * 2;
        int verticesPerLine = (width - 1) / mesh_simplification_increment + 1;

        for (int x=0; x<width;x+=mesh_simplification_increment)
        {
            for (int y = 0; y < height; y+= mesh_simplification_increment)
            {
                data.vertices[vertex_index] = new Vector3(top_left_X+x, height_curve.Evaluate(heightMap[x, y]) * height_multiplier, top_left_Z-y);
                data.UVs[vertex_index] = new Vector2(x / (float)width, y / (float)height);
                if(x<width-1 && y < height - 1)
                {
                    data.AddTriangle(vertex_index, vertex_index + verticesPerLine + 1,vertex_index+ verticesPerLine);
                    data.AddTriangle(vertex_index+ verticesPerLine + 1, vertex_index, vertex_index + 1);
                }
                vertex_index++;
            }
        }
        return data;
    }
}
```

Figura 6.129 Classe “MeshGenerator”

6.14.3 Textura

L’script “TextureGenerator”, mostrat a la figura 6.130, genera una textura2D a partir d’un noise en blanc i negre “HeightMap” o d’un noise passat a color “ColourMap”. La creació d’un noise en blanc i negre serveix únicament per proves i testejos.

```
4 referencias
public static class TextureGenerator
{
    3 referencias
    public static Texture2D TextureFromColourMap(Color[] colourMap,int width, int height)
    {
        Texture2D texture = new Texture2D(width, height);
        texture.filterMode = FilterMode.Point;
        texture.wrapMode = TextureWrapMode.Clamp;
        texture.SetPixels(colourMap);
        texture.Apply();
        return texture;
    }
    2 referencias
    public static Texture2D textureFromHeightMap(float[,] heightMap)
    {
        int width = heightMap.GetLength(0);
        int height = heightMap.GetLength(1);


        Color[] colour_map = new Color[width * height];
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                colour_map[y * width + x] = Color.Lerp(Color.black, Color.white, heightMap[x, y]);
            }
        }
        return TextureFromColourMap(colour_map, width,height);
    }
}
```

Figura 6.130 Script “TextureGenerator”

6.14.4. Visualització

Per poder visualitzar tant el soroll com el mesh generat, utilitzem l'script "MapDisplay", mostrat a la figura 6.131.

```
public class MapDisplay : MonoBehaviour
{
    [SerializeField] private Renderer textureRenderer;
    [SerializeField]
    private MeshFilter mesh_filter;
    [SerializeField]
    private MeshRenderer mesh_renderer;


    // Start is called before the first frame update

    public void DrawTexture(Texture2D texture)
    {


        textureRenderer.sharedMaterial.SetTexture("_BaseMap", texture);
        textureRenderer.transform.localScale = new Vector3(texture.width, 1, texture.height);
    }

    public void DrawMesh(MeshData data, Texture2D texture)
    {
        mesh_filter.sharedMesh = data.CreateMesh();
        mesh_renderer.sharedMaterial.SetTexture("_BaseMap", texture);
        if (mesh_filter.gameObject.GetComponent<MeshCollider>()!=null)
        {
            DestroyImmediate(mesh_filter.gameObject.GetComponent<MeshCollider>());
        }
        mesh_filter.gameObject.AddComponent<MeshCollider>();
    }
}
```

Figura 6.131 Script "MapDisplay"

6.14.5. Color per regions

Per elegir el color que tindrà cada punt del terreny utilitzem regions, mostrat a la figura 6.132. Les regions indiquen el color que tenen els píxels situats en un valor del soroll, o alçada, igual o inferior al valor escollit.

Figura 6.132 Visualització de regions

6.14.6. Implementació

Finalment, utilitzant tots els conceptes explicats previament, generem el mapa utilitzant l'script "MapGenerator". Aquest té 3 funcions de proves: Crear un pla amb la textura del soroll, crear un pla amb la textura del soroll a color i crear un pla amb la textura falloff. A més té la funció que genera el terreny final. (Veure Figures 6.133; 6.134)

```
5 referencias
private enum Drawmode {noise,colour,mesh,falloff}
[SerializeField] private Drawmode draw_mode;

[System.Serializable]
1 referencia
public struct TerrainType
{
    public string name;
    public float height;
    public Color colour;
}

const int mapChunkSize = 121;
[Range(0,6)] [SerializeField] private int LOD;

[SerializeField] private float noise_scale;
[SerializeField] private int octaves;
[SerializeField] private Vector2 offset;
[Range(0,1)]
[SerializeField] private float persistance;
[SerializeField] private float lacunarity;

[SerializeField]
private float mesh_height_multiplier;
[SerializeField]
private AnimationCurve mesh_height_curve;
[SerializeField] private TerrainType[] regions;


[SerializeField]
private bool use_falloff;
[SerializeField]
public bool auto_update;
private float[,] falloff_map;

Mensaje de Unity | 0 referencias
private void Awake()
{
    falloff_map = FalloffGenerator.GenerateFalloffMap(mapChunkSize);
    generate_map();
}
```

Figura 6.133 Inicialització del script "MapGenerator"

```
3 referencias
public void generate_map()
{
    float[,] noise_map = Noise.GenerateNoiseMap(mapChunkSize, mapChunkSize, GameSeed.currentSeed, noise_scale,octaves,persistance,lacunarity, offset);
    Color[] colour_map = new Color[mapChunkSize * mapChunkSize];

    for(int x = 0; x < mapChunkSize; x++)
    {
        for (int y = 0; y < mapChunkSize; y++)
        {
            if (use_falloff)
            {
                noise_map[x, y] = Mathf.Clamp01(noise_map[x, y] - falloff_map[x, y]);
            }
            float currentHeight = noise_map[x, y];
            for(int i=0; i < regions.Length; i++)
            {
                if (currentHeight <= regions[i].height)
                {
                    colour_map[y * mapChunkSize + x] = regions[i].colour;
                    break;
                }
            }
        }
    }

    MapDisplay display = FindObjectOfType<MapDisplay>();
    if (draw_mode == Drawmode.noise)
    {
        display.DrawTexture(TextureGenerator.textureFromHeightMap(noise_map));
    }
    else if (draw_mode == Drawmode.colour)
    {
        display.DrawTexture(TextureGenerator.TextureFromColourMap(colour_map,mapChunkSize,mapChunkSize));
    }
    else if (draw_mode == Drawmode.mesh)
    {
        display.DrawMesh(MeshGenerator.GenerateTerrainMesh(noise_map, mesh_height_multiplier,mesh_height_curve,LOD),
            TextureGenerator.TextureFromColourMap(colour_map, mapChunkSize, mapChunkSize));
    }
    else if (draw_mode == Drawmode.falloff)
    {
        display.DrawTexture(TextureGenerator.textureFromHeightMap(FalloffGenerator.GenerateFalloffMap(mapChunkSize)));
    }
}
Mensaje de Unity | 0 referencias
private void OnValidate()
{
    if (lacunarity < 1)
    {
        lacunarity = 1;
    }
    if (octaves < 0)
    {
        octaves = 0;
    }
    falloff_map = FalloffGenerator.GenerateFalloffMap(mapChunkSize);
}
```

Figura 6.134 Funcions del script “MapGenerator”

6.14.7. Generació de recursos

Per generar recursos aleatoris en el mapa, s’instancien un número específic d’objectes repartits en grups per tot el mapa a l'inici de la partida. (Veure FIgura 6.135)

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class ResourceGenerator : MonoBehaviour
{
    [SerializeField] private float map_size;
    [SerializeField] private int number_of_clusters;

    [SerializeField] private float distance_between_items;

    [SerializeField] private int rocks_per_cluster;
    [SerializeField] private int trees_per_cluster;

    [SerializeField] private GameObject tree_prefab;
    [SerializeField] private GameObject rock_prefab;

    Mensaje de Unity | 0 referencias
    private void Start()
    {
        GenerateResources();
    }
    1 referencia
    public void GenerateResources()
    {
        for(int i = 0; i < number_of_clusters; i++)
        {
            GameObject item_to_spawn;
            int number_of_items;
            if (i % 2 == 0)
            {
                item_to_spawn = tree_prefab;
                number_of_items = rocks_per_cluster;
            }
            else
            {
                item_to_spawn = rock_prefab;
                number_of_items = trees_per_cluster;
            }
            Vector3 position_of_cluster = new Vector3(Random.Range(-map_size / 2, map_size / 2), 100f, Random.Range(-map_size / 2, map_size / 2));
            for(int y = 0; y < number_of_items; y++)
            {
                Instantiate(item_to_spawn, position_of_cluster, Quaternion.identity);

                position_of_cluster = position_of_cluster + new Vector3(Random.Range(distance_between_items,-distance_between_items)+
                    item_to_spawn.GetComponentInChildren<Collider>().bounds.extents.x,0, Random.Range(distance_between_items, -distance_between_items) +
                    item_to_spawn.GetComponentInChildren<Collider>().bounds.extents.x);
            }



        }
    }
}
```

Figura 6.135 Script “ResourceGenerator”

6.15. Extres

A l’hora de generar recursos i enemics en un mapa generat proceduralment hi ha un problema de rotació i alçada, no es pot saber fàcilment com ha d’estar girat l’objecte i a quina posició. Per tal de solucionar aquest problema, tots els objectes que s’instancien s’instanciaran a una alçada molt elevada i s’afegira l’script “Building transform” que reposiciona l’objecte o l’elimina si la posició es inferior a un valor. (Veure Figura 6.136)

```
Script de Unity (9 referencias de recurso) | 0 referencias
public class BuildingTransform : MonoBehaviour
{
    public float threshold=1.5f;
    // Start is called before the first frame update
    Mensaje de Unity | 0 referencias
    void Awake()
    {
        reposition();
    }

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {

    }
    1 referencia
    public void reposition()
    {
        RaycastHit hit;
        Physics.Raycast(transform.position, Vector3.down,out hit, 100000);

        transform.position = hit.point;
        transform.rotation = Quaternion.FromToRotation(transform.up, hit.normal) * transform.rotation;

        if (transform.position.y < threshold)
        {
            Destroy(this.gameObject);
        }
    }
}
```

Figura 6.136 Script “BuildingTransform”

7. Resultat

En aquest apartat analitzarem el resultat final dels objectius del projecte (Veure Apartat 1.2;1.3)
El propòsit del projecte era la creació d'un prototip jugable d'un joc de supervivencia procedural, on el jugador haurà de sobreviure rondes d'enemics aleatoris mentre construeix defenses i les millora amb un sistema de modificació basat en nodes.

Analitzem el nivell d'assoliment de cada objectiu del projecte:

- Generar mapa i recursos procedurals: S'ha creat un sistema de generació de terreny procedural utilitzant textures i soroll per crear un escenari aleatori en cada partida del jugador. Els recursos es generen aleatòriament per tot el mapa.
- Generar ones d'enemics aleatòries: La mecànica s'ha implementat de manera correcta i el sistema funciona com s'havia visualitzat al inici del projecte. Els enemics apareixen utilitzant un sistema de directors que permet una gran complexitat amb pocs recursos.
- Crear un sistema d'inventari: L'estat final de l'inventari és més baix que el visualitzat a l'inici del projecte. No s'ha implementat la mecànica on el jugador pot accedir i manipular els recursos que s'havia pensat en un principi. Malgrat això, el sistema funciona i permet al jugador experimentar amb els materials que obté en la partida.
- Crear un sistema de construcció i millores: El sistema de construcció i mecàniques s'ha implementat com s'havia visualitzat. Les mecàniques de millora i construcció funcionen correctament i estan preparades per ser ampliades en un futur de manera senzilla.
- Expandir el coneixement del llenguatge C# i patrons de disseny de programació: Després de més de mig any treballant en el projecte, ens hem pogut familiaritzar amb els aspectes tècnics de Unity3D, patrons de disseny com l'Observer o el poliformisme i un estil de programació més clar i eficient.

En conclusió, els objectius que s'havien plantejat s'han assolit.

7.1. Captures de Pantalla

Figura 7.1 Menú Principal

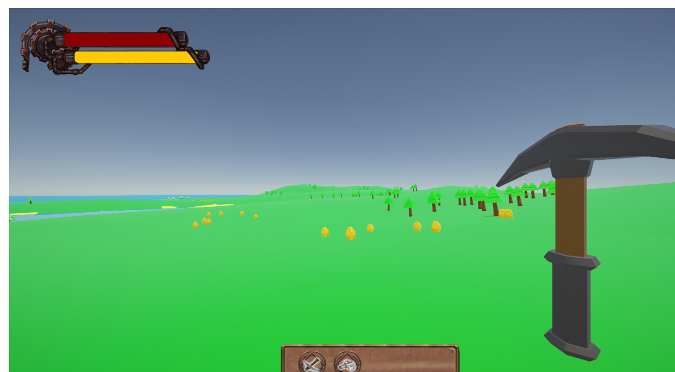

Figura 7.2 Inici de la partida

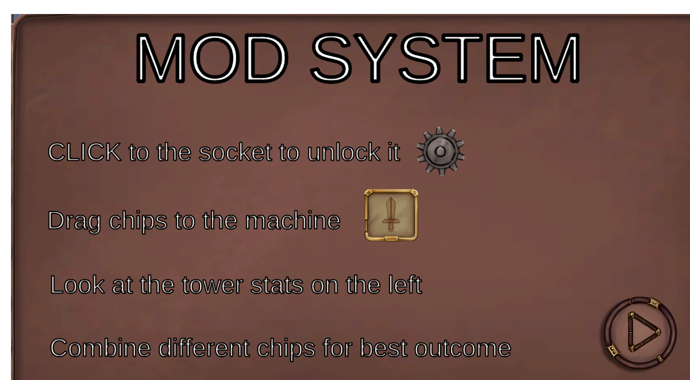

Figura 7.3 Menú d'ajuda part 1

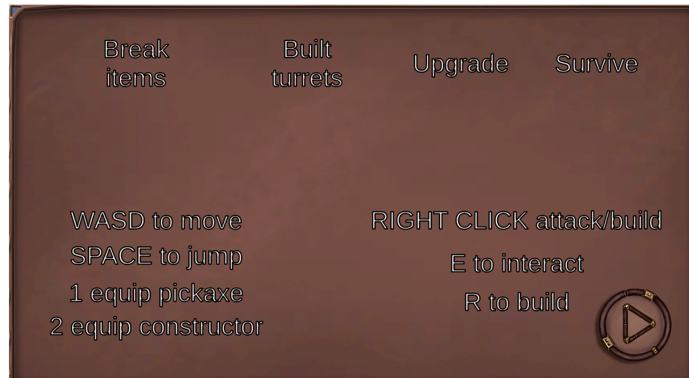

Figura 7.4 Menú d'ajuda part 2

Figura 7.5 Trencant un arbre

Figura 7.6 Construcció d'una torre

Figura 7.7 Estació de millora

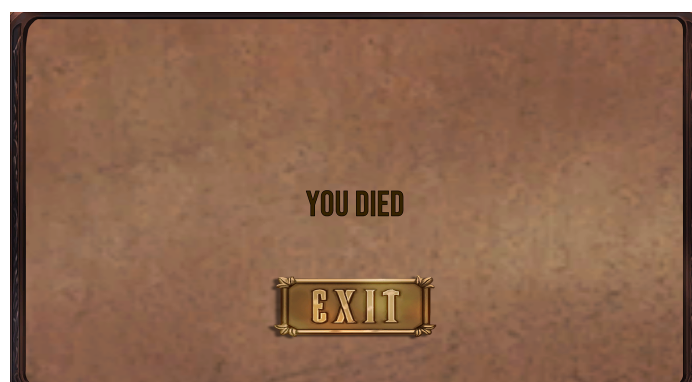

Figura 7.8 Menú de fi del joc

Figura 7.9 Menú d'opcions

8. Conclusions

Aquest projecte ha sigut el projecte més ambiciós que ens hem proposat en l'àmbit dels videojocs. El projecte ha requerit l'ús de totes les habilitats apreses en el curs de Disseny i Desenvolupament de Videojocs. Malgrat la complexitat i el tamany, ha sigut un projecte que s'ha gaudit en tots els aspectes, especialment els tècnics. Ens hem adonat al entrar en l'àmbit del disseny procedural de que hi han moltes coses encara per aprendre. Esperem en un futur crear més projectes que toquin aquest ambit.

A l'inici del projecte, la visió era molt més diferent. El projecte, inicialment, estava enfocat en un videojoc en primera persona de dispars, amb parts de millora i disseny procedural com a extra. Però ens vam adonar que aquesta part proporcionava moltes oportunitats i era poc explorada i l'objectiu va cambiar dràsticament. La idea final va aparèixer al veure el desenvolupament de "Muck", un roguelike procedural i videos d'un creador de contingut anomenat "Sebastian Lague" sobre el tema.

En la fase de disseny, es van crear les parts del projecte amb l'objectiu de dissenyar no un joc sencer, sinó una fundació sòlida amb la que treballar i expandir en un futur les idees plantejades.

Al llarg del projecte han sorgit un gran nombre de errors, majoritàriament a l'apartat de la estació de modificació. Aquests errors han sigut de gran ajuda a l'hora d'aprendre com gestionar correctament matrius i sistemes de nodes.

Finalment, el projecte final conté totes les parts que ens havíem plantejat i les mecàniques estan preparades per ser expandides en un futur.

9. Treball Futur

En aquest apartat explicarem totes les propostes que s’afegirien en el cas d’expandir el projecte en un futur:

- Més opcions de construcció pel jugador, més estructures defensives i més chips de millora.
- Millorar la intel·ligència artificial dels enemics i afegir més varietat.
- Millorar i expandir la generació de terreny i afegir textures i decoració, com per exemple gespa o neu.
- Canviar els models 3D per models de millor qualitat.
- Pulir les mecàniques i afegir noves com objectes especials i armes.
- Afegir més varietat de recursos i un objectiu final pel jugador.
- Millorar la interfície per deixar clar al jugador més aspectes com l’inventari o la hora del dia.
- Afegir so i efectes quan el jugador realitza accions.
- Millorar i modificar la dificultat del videojoc.

10. Bibliografia

Unity Technologies. (2022, Septembre 13). Unity Pro and Unity Enterprise plans: New pricing coming soon. Unity blog. https://blog.unity.com/news/pro-enterprise-new-pricing

Herrero,P. (2023, Maig 1). The 10 best-selling videogames of all time. MeriStation. https://en.as.com/meristation/news/the-10-best-selling-videogames-of-all-time-n/

İSMAIL ÇAMÖNÜ . Raycasting in Unity3D.(2020 Agost 28). CodinBlack. https://www.codinblack.com/raycasting-in-unity3d/

Gentleland. https://www.gentleland.net

Directors. En *Risk of Rain 2 wiki* https://riskofrain2.fandom.com/wiki/Directors

Sebastian Lague. (2016, gener 31) Procedural Landmass Generation (E01: Introduction)[Vídeo]. https://www.youtube.com/watch?v=wbpMiKiSKm8&list=PLFt_AvWsXl0eBW2EiBtl_sxmDtSgZBxB3&index=1

11. Annexos

Com a Annex s’ha adjuntat el projecte sencer de Unity. A dins es pot explorar cada asset utilitzat en el projecte a més de alguns que s’han utilitzat per proves. Tot el codi escrit està en la carpeta situada a “Assets/ScriptsFinal”. Els elements del projecte estan situats per carpetes nombrades segons la funció objectes que conté. A més, s’ha afegit un video d’un fragment d’una partida del joc. Finalment, s’ha ajuntat el prototip executable.

12. Manual d’usuari i instal·lació

12.1 Manual d'instal·lació

El projecte només funciona en ordinadors amb Windows com a sistema operatiu. Per poder iniciar una partida, cal seguir els següents passos:

- Descomprimir l’arxiu ZIP dins d’una carpeta.
- Accedir al directori.
- Executar el fitxer “IslandOfSteel.exe”

12.2 Manual d’usuari

Dins del menú principal, el jugador té l’opció d’accedir a les diferents parts del menú o sortir de la partida. En la partida el jugador pot realitzar les seguents accions:

- Moure’s amb les tecles WASD.
- Saltar amb la tecla espai.
- Canviar d’eines amb les tecles 1 i 2
- Si té el pic equipat pot atacar amb clic esquerra.
- Si té el constructor equipat pot construir una torre amb els recursos requerits amb la tecla “R”.
- Interactuar amb l’estació de millores amb la tecla “E”

A la estació de millores el jugador pot realitzar les següents accions:

- Fer click a les manivelles per activar espais.
- Arrosegar chips a la estació.
- Sortir de l’estació amb la tecla “E”

12.3 Objectiu

L’objectiu del joc és sobreviure a les onades d’enemics que es generen constantment a la nit. Pel dia el jugador pot obtenir recursos i per la nit s'ha de defensar d’enemics cada cop més poderosos.