TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

Generació procedimental de mapes 2D d'estil Metroidvania
MEMÒRIA FINAL

Albert Abulí Federico
Tutor: Rafael González Fernández
2021-2022

TecnoCampus
Mataró-Maresme

Abstract

The goal of this project is to develop a 2D Metroidvania-style map generator that meets a number of rules and requirements and is generated procedurally as Roguelike games do. In order to achieve this purpose, other games of the same genre and various procedural generation techniques will be analyzed, which will later be used to carry out the tool using Unity.

Resum

L’objectiu d’aquest projecte és desenvolupar un generador de mapes 2D de l'estil Metroidvania que compleixi una sèrie de normes i requisits de i que es generi de forma procedimental tal com els jocs del gènere Roguelike ho fan. Per tal d’aconseguir aquest propòsit, s’analitzaran altres jocs del mateix gènere i diverses tècniques de generació procedimental, que a posteriori es faran servir per dur a terme l'eina utilitzant Unity.

Resumen

El objetivo de este proyecto es desarrollar un generador de mapas 2D del estilo Metroidvania que cumpla una serie de normas y requisitos de y que se genere de forma procedimental, tal y como lo hacen los juegos del género Roguelike. Para conseguir este propósito, se analizarán otros juegos del mismo género y diversas técnicas de generación procedimental, que a posteriori se utilizarán para llevar a cabo la herramienta utilizando Unity.

Índex

Índex de figures

Índex de taules

1. Introducció

Actualment, podem trobar un munt de gèneres o etiquetes que classifiquen els videojocs segons la seva forma de ser, com es juguen o com es veuen. Dins dels jocs 2d hi ha dos d'aquests gèneres que són molt famosos i que tenen una gran quantitat de fans i desenvolupadors al darrere, aquests dos són els Metroidvania i els Roguelike.

Tant els Roguelike com els Metroidvania tenen bastant en comú, com per exemple l'estètica 2d o l'opció de poder millorar el personatge principal a mesura que avances la partida. El punt més important a destacar, però, és l'exploració, i és que en els dos gèneres es podria classificar com un dels eixos o pilars centrals del seu *gameplay*. La Diferencia, tanmateix, és que els Metroidvania són pensats i dissenyats per guiar al jugador a través del mapa seguint una ruta mentre que els Roguelike no.

Els Metroidvania estan dissenyats de tal manera que el jugador sempre es toparà amb alguna porta o entrebanc que bloquejarà el seu camí, per aquest motiu, haurà d'explorar el mapa per tal d'aconseguir claus, *items* o habilitats noves que li permetin superar els obstacles que s'ha trobat prèviament. En contrapart, els Roguelike són masmorres generades de forma procedimental, i per tant, el mapa, els enemics o les recompenses sempre seran diferents a cada partida a causa de l'aleatorietat del generador.

En l'actualitat podem trobar centenars de jocs d'aquest d'aquests dos estils, i no és d'estranyar, ja que són molt populars. Alguns exemples moderns són: la saga de Ori (Moon Studios GmbH, 2015-2020), Hollow Knight (Team Cherry, 2017), Blasphemous (The Game Kitchen, 2019), The Binding of Isaac (McMillen i Himsl, 2011) o Hades (Supergiant Games, 2020).

Últimament, però, sembla que hi ha una petita tendència d'intentar implementar un sistema de generació procedimental en els mapes Metroidvania de la mateixa forma que ho fan els jocs Roguelike. Videojocs com ara: el Bloodstained: Ritual of the Night (ArtPlay, 2019), Sundered (Thunder Lotus Games, 2017), A

Robot Named Fight! (Morningstar Game Studio, 2017) o Chasm (Bit Kid, Inc., 2018) han intentat fusionar aquests dos generes per tal de poder aconseguir-ne els beneficis sense perdre'n l'essència.

Així doncs, aquest treball se centrarà a investigar i estudiar formes de desenvolupar un generador de mapes 2d de l'estil Metroidvania de manera procedimental. Aquest generador de mapes estarà dut a terme en Unity com a motor de desenvolupament.

2. Antecedents de la recerca/Referents

La generació procedimental és una eina molt potent que fa molts anys que es porta fent servir. Tanmateix, fa relativament poc que s'intenta aplicar en jocs d'estil Metroidvania. En podem veure uns exemples com a referents:

2.1. Chasm (Bit Kid, Inc., 2018)

Chasm és un videojoc desenvolupat per Bit Kid, Inc. i llençat el 2018. Es defineix com *action-adventure* d'estil Metroidvania i un dels seus punts a destacar és que les sis àrees que conté dins del joc són generades de forma procedimental.

Figura 1. Gameplay de Chasm. Font: Bit Kid Inc., 2018.

James Petruzzi, director de Bit Kid Inc., detalla en un blog de Playstation el motiu darrere la generació procedimental en comptes d'un disseny de mapa convencional. Argumenta que els Metroidvania tradicionals un cop completats perden encant, i és que quan el jugador ja ha explorat el mapa prèviament, ja sap a on és cada arma, poder o enemic. Per aquest motiu, explica que la re-jugabilitat d'aquests tipus de videojocs es veu afectada negativament, i que l'element d'exploració es perd. Per tal d'arreglar això, la seva proposta ha estat fer un Metroidvania procedimental per fer que cada partida se senti diferent i refrescant pel jugador. (Petruzzi, 2018).

Seguidament, detalla el procés en detall que van seguir a l'hora de generar les sis zones de forma procedimental. Petruzzi explica que no tot el mapa és generat de forma aleatòria, les parts estructurals fonamentals són sempre fixes, mentre que els passadissos que les connecten són els elements que es generen de forma procedimental. A continuació, exposa una imatge per representar el mètode explicat:

Figura 2. Sales estructurals i connexions procedimentals de Chasm. Font: Bit Kid Inc., 2018.

Un altre detall a tenir en compte del videojoc Chasm és que les habitacions són dissenyades a mà i guardades en una llibreria, que més endavant, el generador utilitza per generar el camí que uneix les sales estructurals immòbils tal com Petruzzi explica.

Per últim, les sis zones que conformen el mapa del joc estan separades una de l'altre evitant d'aquesta manera els possibles problemes de col·lisió entre elles quan es generen:

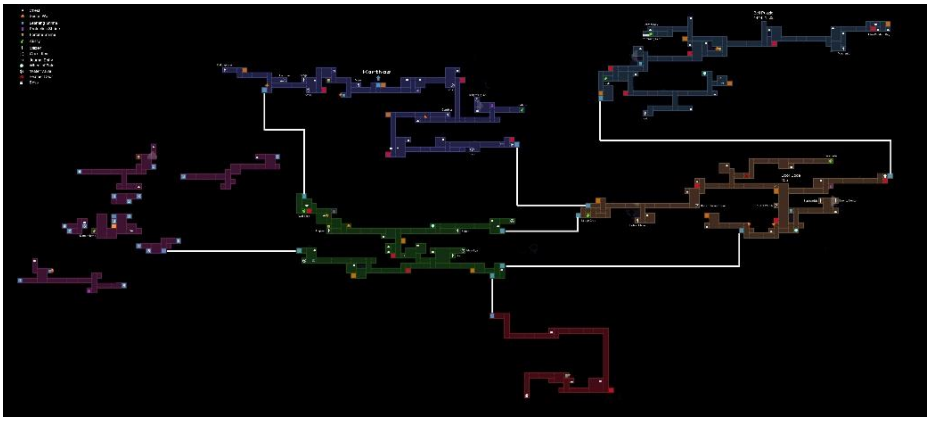

Figura 3. Mapa de Chasm. Font: BitKid Inc., 2018.

2.2. Bloodstained: Ritual of the Night (ArtPlay, 2019)

Bloodstained és un joc desenvolupat per ArtPlay i llençat durant el 2019. El que fa especial aquest joc és que la persona encarregada del projecte és Koji Igarashi, antic productor de la saga Castlevania de Konami. El videojoc va començar la seva campanya a través de Kickstarter, una web que promociona idees o projectes per tal que la gent poguí donar suport monetari amb la finalitat que el projecte pugui ser dut a terme.

Un dels objectius del joc segons Kickstarter era la inclusió d'un mode per generar mapes de masmorres produïts de forma procedimental tal com fan els jocs Roguelike. Malauradament, aquest mode no va ser possible. Igarashi va fer un comunicat explicant el motiu darrere d'aquesta notícia, i és que, el codi que havien desenvolupat al principi del projecte per la generació procedimental del castell no era compatible amb el joc final. (Igarashi, 2020).

Tot i això, es va implementar un sistema anomenat Randomizer com a substitut. Aquesta eina permet als jugadors modificar els paràmetres de la partida com el mapa, els enemics o els ítems per tal que siguin totalment aleatoris. Amb aquesta funció afegida, el jugador pot assolir una experiència diferent i refrescant cada cop que inicií una partida, tot i que el mapa seguirà sent el mateix amb excepció d'algunes sales.

Figura 4. Randomizer del joc Bloodstained Ritual of the Night. Font: ArtPlay, 2019.

2.3. A Robot Named Fight! (Morningstar Game Studio, 2017)

A Robot Named Fight! és un joc desenvolupat per Morningstar Game Studio durant l'any 2017. Es defineix com un joc Metroidvania-Roguelike a on l'exploració i els elements col·leccionables són els seus pilars principals. De la mateixa manera que els Roguelike, el mapa del joc es genera de forma procedimental i inclou la característica del *permadeath*, que fa que si el jugador mor, ha de tornar a començar la partida.

Figura 5. Gameplay de A Robot Named Fight!. Font: Morningstar Game Studio, 2017.

En un Q&A que va tenir lloc durant el 2018 a la plataforma de Reddit el desenvolupador del joc, Matt Bitner, explicava el motiu del perquè havia decidit fusionar aquests dos generes en un mateix joc. Segons Bitner hi ha molt de potencial en els jocs que es munten intel·ligentment mitjançant algorismes i decisions aleatòries i, per tant, va decidir desenvolupar un videojoc similar a un Metroid amb aquestes característiques. Bitner afirma que semblava la decisió més encertada per moure el gènere endavant. (Bitner, 2018)

En el mateix Q&A Matt Bitner també explica la complexitat del generador de mapes i les complicacions que es va trobar a l'hora d'escriure l'algorisme sense tenir cap mena de coneixement previ en la matèria. Especifica que les habitacions

són dissenyades a mà i després són col·locades en el mapa de manera aleatòria pel generador.

3. Objectius:

En aquesta secció del treball s'explicaran els diferents objectius proposats per dur a terme durant el desenvolupament del projecte. Els objectius s'han separat en dos tipus, els principals i els secundaris.

3.1. Objectius principals

- Desenvolupar un generador de mapes 2D funcional d'estil Metroidvania. L'eina haurà de ser capaç de generar diversos tipus de mapes de manera procedimental depenent dels paràmetres que rep. A part, haurà d'incloure els següents elements clau del gènere Metroidvania:
    - Diverses zones.
    - Mapa interconnectat.
    - Sales de guardar partida i sales de teleportació.
    - Sales de transició entre zones.
    - Sales de porta i sales clau.
    - Sales de *boss*.
- Fer que el generador sigui una eina fàcil de fer servir, intuïtiva i modificable utilitzant editors personalitzats i noves finestres o menús per facilitar la interacció de l'editor de Unity.

3.2. Objectius secundaris

- Fer que els mapes generats siguin jugables per tal de provar-los.
- Poder exportar o importar els mapes generats.
- Implementar més opcions de personalització o sales especials.

4. Marc teòric

L'objectiu principal d'aquest treball és desenvolupar un generador de mapes procedimentals 2D d'estil Metroidvania, per tant, s'explicarà l'origen d'aquest gènere de videojocs, les característiques que tenen i s'analitzaran exemples d'altres jocs del mateix estil. Altres conceptes a destacar en aquest projecte són el gènere de videojocs anomenat Roguelike i les tècniques de generació procedimental conjuntament amb els algorismes que utilitzen.

S'analitzaran diferents tipus d'algorismes de generació procedimental i s'explicarà com implementar-los, el seu funcionament i en què podrien beneficiar aquest projecte.

4.1. Roguelike

Roguelike és un subgènere del gènere RPG popularitzat durant els anys vuitanta. El nom d'aquest gènere prové d'un videojoc anomenat Rogue (Toy i Wichman, 1980). Aquest videojoc és conegut per l'ús de generació procedimental de nivells i l'ús de caràcters ASCII per representar un sistema de til·les que confeccionaven el mapa de les catacumbes (Watkins, 2016).

En el llibre Exploring Roguelike Games (Harris, 2021), els jocs Roguelike són descrits com a jocs d'exploració de masmorres ambientats en mons generats aleatòriament mitjançant diversos algorismes o tècniques de generació procedimental. Són coneguts per la seva alta dificultat, per ser imprevisibles, tenir la característica de mort permanent del personatge i una gran varietat de formes de matar al jugador com per exemple amb l'ús de trampes, emboscades d'enemics o recursos limitats.

Alguns exemples de jocs Roguelike són The Binding of Isaac (McMillen i Himsl, 2011), Dead Cells (Motion Twin, 2018) o Hades (Supergiant Games, 2020).

4.2. Metroidvania

Metroidvania és un tipus de gènera de videojocs que neix i es consolida durant els anys noranta amb el llançament de dos títols que van marcar un abans i un després, es tracten del Super Metroid (Nintendo, 1994) i el Castlevania Symphony

of the Night (Konami, 1997). El nom del gènere, Metroidvania, està compost per la unió dels seus dos noms. A part del nom, molts dels elements clau que constitueixen el gènere avui dia també van ser heretats d'aquestes dues grans sagues. (Minkkinen, 2016).

El gènere està centrat en l'exploració, però incorpora elements d'altres generes com plataformes o rpg. A més, el Metroidvania es caracteritza per incorporar un sistema de *gating* o sistema de portes. Tot i el nom que té aquest sistema, no té per què ser una porta literalment, essencialment es tracta d'una barrera que bloqueja el progrés del jugador fins que aquest aconsegueixi l'ítem o l'habilitat necessària per obrir-la (Stalnaker, 2020).

Alguns exemples d'aquest gènere són el Hollow Knight (Team Cherry, 2017), Blasphemous (The Game Kitchen, 2019) o Bloodstained: Ritual of the Night (ArtPlay, 2019).

4.2.1. Metroid

Metroid (Nintendo, 1986-2021) és una saga de jocs que originalment va fer la seva primera aparició al Japó l'any 1986 i va ser distribuït per la Family Computer (Famicom) de Nintendo. Un any després, el 1987, va ser llençat a Nord Amèrica per la Nintendo Entertainment System (NES).

El joc va ser publicat per Nintendo, produït per Gunpei Yokei i desenvolupat per Nintendo R&D1(Nintendo Research & Development) i Intelligent Systems. Va ser un joc 2D molt innovador a on va ser dels primers jocs a incorporar un sistema de contrasenyes que permetia al jugador guardar la partida o obtenir avantatges com per exemple power-ups, munició o l'opció de treure l'armadura a la protagonista (Hanson, 2018, p.97). També va ser dels primers a incorporar un mapa obert a on es podia tornar al punt d'inici o accedir a zones prèviament bloquejades en comptes de tenir nivells com altres jocs desenvolupats per la mateixa companyia japonesa. (Bissell, 2010, p. 97)

L'últim llançament d'aquesta saga, Metroid Dread, va ser llençat l'any 2021 i dut a terme per Mercury Steam, una empresa desenvolupadora de jocs espanyola

que també va treballar en alguns títols de la saga de Castlevania. Aquesta última entrega de Metroid ha venut més de 2,9 milions de còpies a escala mundial.

4.2.2. Castlevania

Castlevania (Konami, 1986-2019) és una saga de videojocs a on el seu primer títol també va ser llençat l'any 1986 al Japó per la Famicom i va ser publicat i desenvolupat per Konami. De la mateixa manera que el Metroid, el primer Castlevania també era un joc 2D, però amb la principal diferència de tenir el mapa dividit en nivells en comptes de ser completament obert. Més tard adoptarien el mapa obert igual que Metroid amb la incorporació d'elements RPG que fomentaria l'estil icònic de la saga.

Precisament, el joc més icònic de la saga és el Castlevania Symphony of the Night (1997), que va incorporar elements de RPG (Bycer, 2018, p.7), com per exemple la varietat d'armes que el jugador es podia equipar, nivells, power-ups de personatge o les màgies secretes que es podien desbloquejar si el jugador introduïa una seqüència de moviments i accions específiques. A més, la fluïdesa de controls va ser molt superior en vers les entregues anteriors. Moltes d'aquestes mecàniques han sigut heretades per les entregues que van seguir aquest joc posteriorment. (Kalata, 2017)

L'últim llançament d'aquesta saga, Castlevania Advance Collection, va ser publicat l'any 2021 per diverses plataformes. Es tracta d'una recopilació i port d'anteriors títols a les plataformes actuals.

4.2.3. Elements clau d'un Metroidvania

Així doncs, Metroidvania és un gènere que classifica tots aquells jocs que han agafat elements clau d'aquestes dues sagues i que han incorporat una sèrie de normes que són necessàries per a distingir-se de tots aquells jocs que són similars a primera vista, com els jocs d'acció 2D o jocs de plataformes, però que no són Metroidvania.

Segons Cossu (2019) un videojoc que forma part del gènere Metroidvania té les següents particularitats:

- Són jocs no lineals, de tal manera que no tenen nivells o un únic camí delimitat per un inici i un final i que fomenten l'exploració. Tot i que els mapes del gènere Metroidvania normalment sí que estan dividits en zones que tenen una certa similitud amb els jocs de nivells, el mapa és un únic escenari a on el jugador té total llibertat per poder explorar i recular quan ell ho desitgi.

Podem veure un exemple d'això si es comparen els mapes dels jocs Super Metroid i Super Ghouls 'n Ghosts (Capcom, 1991). El mapa del Super Metroid [Figura 6] és molt més laberíntic i està més interconnectat. En el segon mapa, el del Super Ghouls 'n Ghost [Figura 7], podem veure un mapa molt lineal a on cada segment d'aquest és un nivell diferent amb un principi i un final, per tant, evita que el jugador tingui l'opció a retrocedir a segments anteriors.

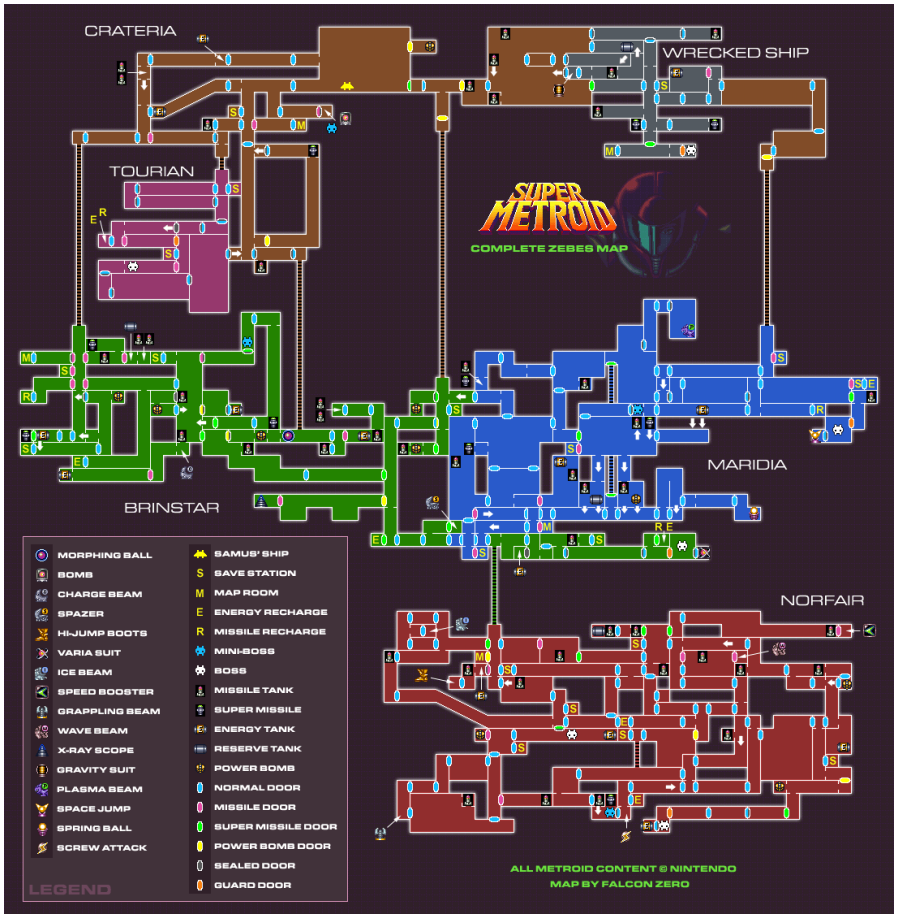

Figura 6. Mapa de Super Metroid. Font: Nintendo, 1994.

Figura 7. Mapa de Super Ghouls 'n Ghosts. Font: Capcom, 1991.

- Adquisició d'*items* o habilitats que permetran al jugador accedir a àrees noves a on prèviament eren inaccessibles. (Cossu, 2019).

  Un exemples d'això, es pot veure en els jocs de Metroid, a on el jugador ha d'aconseguir l'habilitat anomenada *morphing ball,* aquesta permet al jugador transformar-se en una bola petita que li permetrà passar pels forats que es troba al llarg de tot el joc o el doble salt que li permetrà accedir a llocs més elevats.

4.3. Característiques d'un mapa d'estil Metroidvania

Segons Gutiérrez Rodríguez et al., els mapes del gènere Metroidvania són grans i interconnectats a on el jugador es pot moure, ha d'aconseguir objectes, armes o habilitats per desbloquejar les diferents àrees que estan bloquejades. El mapa està compost de diferents àrees i cada una d'aquestes està composta de diverses habitacions, algunes d'aquestes amb característiques especials com les sales de transició, sales de guardar partida o les sales secretes. Dins de les habitacions serà on els diferents enemics, objectes o noves habilitats són posicionades. (Gutiérrez Rodríguez et al., 2018).

Tanmateix, per tal d'entendre què diferencia a un mapa d'estil Metroidvania de la resta, s'ha agafat com a exemple el mapa del Castlevania Symphony of the Night per analitzar-lo i documentar els punts més rellevants. Aquests punts després serviran de guia per replicar-los en el generador de mapes 2D desenvolupat en aquest projecte amb Unity.

4.3.1. Zones del mapa

Com s'ha mencionat anteriorment, tot i que els jocs de l'estil Metroidvania no tenen nivells, el mapa que tenen està dividit en zones que simulen els diferents nivells que podem trobar a altres jocs, ja que cada nivell acostuma a tenir una temàtica pròpia a on l'art, els enemics o les mecàniques que contenen són exclusives a aquella zona.

Per posar un exemple d'això, podem veure la imatge del mapa del Castlevania Symphony of the Night, [Figura 8], a on les diferents zones estan pintades de diferents colors i cada una és totalment diferent de les altres temàticament.

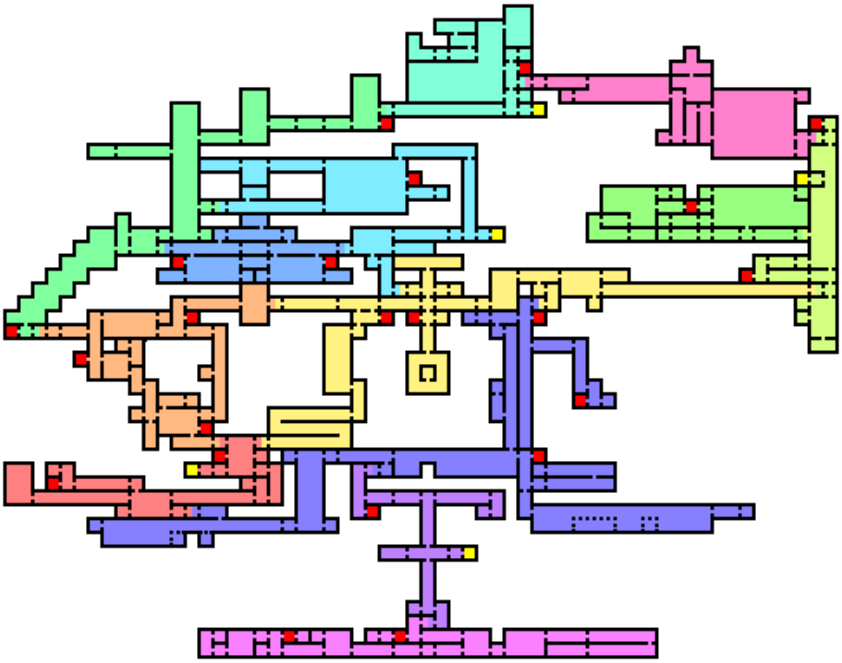

Figura 8. Mapa i zones del videojoc Castlevania Symphony of the Night. Font: Konami, 1997.

4.3.2. Connexions de zones

Generalment, cada zona està comunicada amb algun altre, amb una sola connexió o amb diverses a la vegada. Si la zona només té una sola connexió, aquesta mateixa serà l'entrada i la sortida, mentre que si la zona té diverses connexions, la seva funció serà de *hub* i el jugador podrà accedir a les altres zones des d'allà.

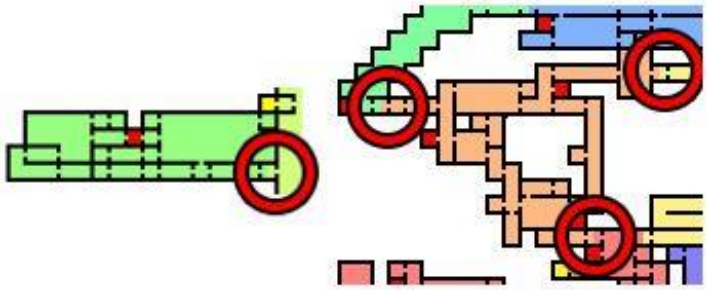

Figura 9. Exemple de conexions entre zones del videojoc Castlevania Symphony of the Night. Font: Konami, 1997.

4.3.3. Zones inaccessibles

Tal com s'ha mencionat abans, aquest és un dels punts clau de qualsevol Metroidvania. El jugador es toparà constantment amb portes inaccessibles o entrebancs que no podrà superar fins a aconseguir una habilitat o un *item* en concret.

Normalment, aquestes *gate rooms* o sales de porta, són sales que el jugador haurà de recordar per tornar-hi en un futur, per tant, s'acostumen a dissenyar amb elements que ressalten a la vista i que són fàcils de recordar.

Figura 10. Habitacions amb obstacles del videojoc Castlevania Symphony of the Night. Font: Konami, 1997.

4.3.4. Sales especials comunes

Tot i que el mapa està format per zones amb diferents temàtiques, hi ha algunes sales que són especials i que totes les zones comparteixen, aquestes habitacions que tenen en comú, són les sales especials amb mecàniques específiques. Generalment, es pot trobar tres tipus de sales comunes, aquestes són:

- Sales per guardar la partida.
- Sales de teleportació.
- Sales de transició entre zones.

En ser especials, aquestes sales han de seguir una sèrie de normes:

- **Número de sales de guardar:** Les habitacions de guardar són proporcionals a la mida de la zona en la qual es troben. És a dir, si una zona és molt petita, és possible que només tingui una sola sala de guardar, mentre que si és molt gran, en tindrà diverses.
- **Posició de les sales de guardar:** Sempre es posarà una sala de guardar a prop d'una sala d'enemic o *boss* final de zona. Aquestes sales són situades allà en cas que el jugador no superi el repte i així tenir l'oportunitat de tornar a intentar-ho cada cop que siguin necessaris sense haver de tornar a explorar una zona. Si la zona és gran, la resta de sales de guardar es repartiran dins d'aquesta.
- **Sales de teleportació:** De la mateixa manera que les sales de guardar, aquestes sales estan escampades per les zones del mapa, la principal diferència però, és que com a màxim només hi pot haver una sala d'aquest tipus per zona, a més, es pot donar el cas d'haver-hi zones sense sala de teleportació, ja que són opcionals. Normalment, es posen a les zones més transitades a on el jugador haurà de fer més *backtraking*.
- **Sales de transició:** Aquestes sales només apareixen entre zona i zona. No tenen *items* ni enemics i la seva principal funció és comunicar al jugador que accedirà a una zona diferent de la que es trobava.

4.3.5. Sales secretes

A vegades hi ha habitacions secretes que no apareixen al mapa fins a ser descobertes. Aquestes normalment tindran algun tipus de recompensa a dins. Per obrir aquestes sales el jugador haurà de destruir la falsa paret que la tapa o activar algun tipus de mecanisme amagat per després poder accedir-hi.

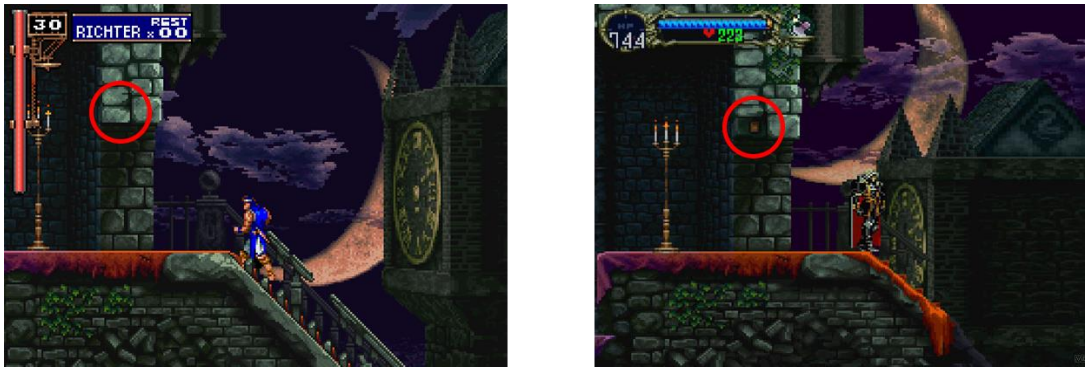

Figura 11. Exemple d'interruptor amagat per obrir una sala secreta. Font : Konami, 1997.

4.3.6. Mida i forma de les habitacions

Les habitacions acostumen a ser grans i en forma quadrada o rectangular. Generalment, la mida de les habitacions acostumen a ser mitjanes o grans, així que les habitacions d'una única cel·la són menys freqüents que la resta i normalment són reservades per habitacions que contenen premis pel jugador o habitacions especials com les sales de guardar, sales de teleportació, sales secretes i sales de canvi de zona.

4.4. Procedural Generation (PCG)

La generació procedimental té molt de pes en aquest projecte, per aquest motiu, és important establir el seu significat abans d'entrar en detall de les tècniques o especificitats.

Rayan Watkins (2016) defineix la paraula procedure, dins del món de la programació, com a una instrucció a executar amb la finalitat de comunicar a l'ordinador el que volem que faci o completi. (Watkins, 2016, p.48).

Si busquem la definició completa, Roland van der Linden, Ricardo Lopes i Rafael Bidarra (2014) defineixen la generació procedimental com a la creació algorítmica de contingut generat de forma automàtica. (van der Linden et al., 2014)

Un altre exemple de la definició de PCG la podem trobar en el llibre anomenat Procedural Content Generation in Games escrit pels autors Noor Shaker, Julian

Togelius i Mark J. Nelson (2016). En aquest llibre descriuen la generació procedimental com a la creació algorítmica de contingut amb una interacció mínima o limitada de l'usuari (Shaker, N., Togelius, J., & Nelson, M. J., 2016).

Així doncs, gràcies a la generació procedimental es pot fer que l'ordinador s'encarregui de generar contingut automàticament que, d'altra banda, una persona hauria de desenvolupar o dissenyar.

La generació procedimental es porta aplicant a diverses indústries des de fa bastants anys, en el cinema per exemple, s'utilitza per generar ciutats, terrenys, boscos o exèrcits de soldats com a les pel·lícules de la trilogia de The Lord of the Rings (2001 - 2003) dirigides per Peter Jackson (Moana Thompson, 2006). En els videojocs també fa temps que es fa servir la generació procedimental, alguns exemples de jocs són el Dead Cells (2018, Motion Twin) o el The Binding of Isaac (2011), que fan ús del PCG a l'hora de generar els mapes que el jugador haurà de travessar.

4.4.1. Propietats de la generació procedimental

Quan es treballa amb la generació procedimental es busquen unes característiques o propietats en concret que determinaran la qualitat de l'eina desenvolupada o la seva usabilitat. Noor Shaker, Julian Togelius i Mark J. Nelson (2016) i expliquen en el seu llibre les cinc propietats o qualitats que són desitjades quan es treballa amb PCG:

- **Velocitat**: El temps que es tarda o la velocitat en la qual s'executa el programa o el joc per generar el contingut. Aquesta propietat depèn de les altres, però en general es busca aconseguir una velocitat elevada.
- **Fiabilitat**: Propietat que determina si el resultat obtingut sempre compleix una sèrie de paràmetres o criteris en concret.
- **Controlabilitat**: Determina si l'usuari té l'opció de controlar o gestionar paràmetres o valors del programa per assolir un resultat determinat.

- **Expressivitat i diversitat**: Qualitat que determina si el resultat generat és avorrit o repetitiu.

- **Creativitat i credibilitat**: Propietat que determina si el resultat obtingut pot ser distingit entre contingut generat manualment i contingut generat automàticament.

A vegades no es pot aconseguir tenir totes les qualitats desitjades a la vegada i acaba convertint-se en un intercanvi relatiu entre elles. Depenent de cada projecte es posarà més pes en algunes àrees que en altres, si per exemple es busca velocitat per sobre de tot, pot ser que la diversitat o la credibilitat es vegin afectades o viceversa.

4.4.2. Tipus de PCG en els videojocs

Com s'ha mencionat anteriorment, la generació procedimental fa molts anys que es porta aplicant a diferents tipus d'indústries, però en videojocs, Watkins (2016) especifica que tot i que el PCG es pot fer servir en gairebé tots els àmbits que formen un videojoc, es pot classificar en set categories diferents:

Tipus	Descripció	Exemple de videojoc que utilitza el tipus de PCG
Maquetació de nivell	Generació de nivells o mapes.	Diablo (Blizzard Entertainment, 1997)
Creació d'ítems	Generació d'ítems o armes.	Borderlands (GearBox Software, 2009)
Comportament de la IA	El comportament de la IA dels NPCs.	Dreeps: Alarm Playing Game (Hiraoka & Watanabe, 2015)

Generació de textures	Generació de textures.	.kkrieger (.theprodukkt, 2004)
Generació de models.	Generació de models, com per exemple vegetació o personatges.	SpeedTree (Interactive Data Visualization, Inc., 2002)
Trama argumental	Generació de la història o sistema de missions.	The Elder Scrolls 5: Skyrim (Bethesda Game Studios, 2011) – Radiant Quests System.
Música	Generació de música o efectes sonors.	Proteus (Key & Kanaga, 2013)

Taula 1. Tipus de PCG. Font: Watkins, 2016.

4.4.3. Inconvenients o riscos de fer servir PCG

Tot i que la generació procedimental pot semblar una eina molt potent i útil, també ho són els inconvenients que genera. Grey ho explica de la següent manera:

> *PCG comes with a warning label. It can be a black hole, a slippery slope, a project risk, and a dark abyss. It's important to understand these risks when looking to integrate PCG into your game, and to realize all the implications and impacts of PCG across your project.* (Grey, 2017, p.5)

Alguns d'aquests riscos poden ser decisius a l'hora de decidir si utilitzar PCG en un projecte o no. Segons Grey (2017) els problemes més habituals són la falta de temps a causa d'invertir-lo en el desenvolupament de l'eina en comptes del videojoc, ja que el generador s'ha de construir, modificar i afinar a les necessitats del projecte, sobrepassar el límit de capital del projecte, possibles *bugs*, una gestió

de balanceig incorrecte o una falta de qualitat dels possibles resultats que es poden obtenir del generador.

Per aquest motiu és vital preparar-se i organitzar-se per tal d'aconseguir una base sòlida amb la qual poder treballar. Un dels punts claus per construir aquesta base són els algorismes que necessitarem per desenvolupar el generador.

4.5. Algorismes de mapa

Dins del món de la generació procedimental ens podem trobar un munt d'algorismes i tècniques de generació, tot i això, no tots els mètodes són adients per totes les feines, per aquest motiu, en el cas d'aquest treball s'han seleccionat un nombre d'algorismes que estan alineats amb l'estil de mapa que es vol generar.

4.5.1. Random Room Placement

Segons Baron (2017), aquest mètode és un algorisme de força bruta per la generació d'habitacions. S'implementa generant habitacions de mides aleatòries que després s'intenten col·locar a punts aleatoris de la quadrícula, en cas que l'habitació no col·lideixi amb cap altre i estigui dins de la graella es pintarà. Aquesta acció es repetirà fins que s'obtingui el número d'habitacions desitjat o la graella tingui un percentatge ocupat superior al valor establert. Aquestes habitacions posteriorment s'hauran d'unir mitjançant passadissos en cas que hi hagi espai entre elles.

D'altra banda, Himsl (2020) explica com va fer servir una tècnica similar pel desenvolupament de The Binding of Isaac (2011) però amb les diferències que en comptes de generar les habitacions d'una mida aleatòria, ell va fer servir plantilles de sales d'una mida i forma preestablertes i que en comptes de seleccionar un lloc aleatori de la quadrícula, ell utilitzava les portes de les habitacions anteriors com a lloc de *spawn* de les noves.

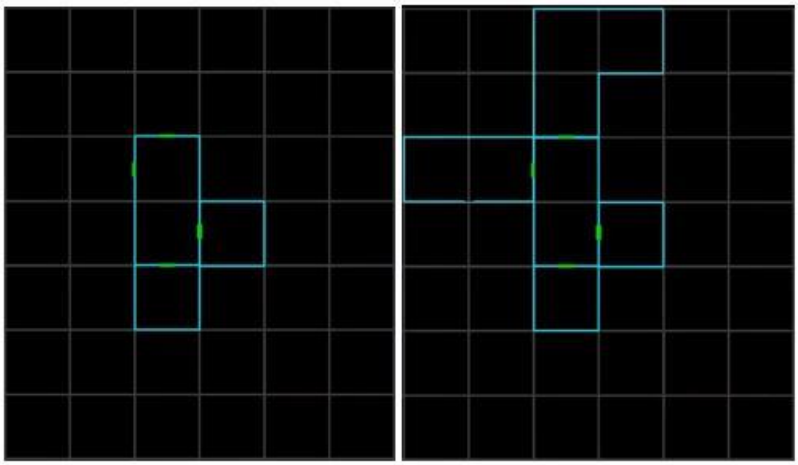

Figura 12. Exemple de Random Room Placement. Font: Himsl, 2020.

El procés de l'algorisme és el següent:

1. **Seleccionar una sala:** En aquest pas, s'agafarà una de les plantilles de sales o es generarà una habitació amb una mida aleatòria.

2. **Col·locació de la sala:** Un cop seleccionada la sala, s'intenta col·locar dins de la graella a un punt aleatori o al darrere de l'habitació anterior. Si el lloc a on es vol col·locar la sala ja està ocupat o està fora de la graella, la sala es descarta.

3. **Comprovació:** Es comprova si el nombre de sales col·locades ha arribat al mínim especificat, si és així, l'algorisme pararà la seva execució donant per acabat el mapa, en cas contrari, el cicle torna a començar.

L'últim pas, en cas de fer ús de plantilles com Himsl, és substituir aquestes sales plantilla per habitacions predissenyades que compleixin les característiques que les plantilles.

Les característiques són les següents:

- El nombre de portes obertes.

- La posició de les portes que té.

- La mida i la forma de la sala.
- El contingut de l'habitació o el tipus de sala.

A més si les habitacions no estan unides per portes com en el cas de The Binding of Isaac, s'hauran d'unir mitjançant passadissos tal com s'ha mencionat en el cas de Baron.

Tot i ser força senzill aquest algorisme, és poc eficient i és necessari tenir plantilles i una gran llibreria d'habitacions predissenyades ens en cas de seguir l'exemple de Himsl, però per contra es té molt de control de les habitacions.

4.5.2. Drunkard's Walk o Random Walker

Aquest algorisme és un dels més coneguts i utilitzats per crear coves o mapes a on totes les sales estan connectades per un camí com a mínim. L'algorisme s'encarrega de posar un agent una casella d'una graella, la feina d'aquest agent, és moure's cap a una direcció aleatòria de les que pot seleccionar. Obté el nom per la manera de com es mou l'agent. Aquest algorisme pararà quan els agents hagin superat el màxim de passos permesos o arribat a la meta establerta.

Bucklew (2017) explica que alguns dels principals usos d'aquest tipus d'algorisme és per generar camins per connectar dos punts d'interès, crear un sistema de cavernes o crear una xarxa similar a un clavegueram o de rius. Afegeix que és molt comú descartar els camins que no són útils al final de la generació.

El procés de l'algorisme és el següent:

1. **Delimitar els moviments de l'agent:** Depenent del resultat que es vulgui aconseguir, es delimitarà el moviment de l'agent. Per exemple es pot especificar que només es pugui moure en dues direccions, quatre o vuit. És possible modificar els moviments de l'agent en el transcurs de l'operació per tal d'evitar escenaris no desitjats, com per exemple passar per un lloc que ja s'ha visitat o sortir dels límits de la graella.

2. **Col·locar l'agent a la graella:** Un cop es té l'agent, aquest serà col·locat a una casella de la graella. És possible col·locar-ne més d'un per assolir un mapa més complex i gran.

3. **Seleccionar un moviment:** Se seleccionarà aleatòriament un dels moviments delimitats de l'agent.

4. **Executar el moviment:** L'agent es mourà cap a la direcció seleccionada.

5. **Comprovació:** Es repetirà el pas 3 i 4 fins que s'arribi a un nombre desitjat de sales visitades o s'arribi a la meta establerta.

En el cas que vulguem arribar a un punt concret del mapa, és possible modificar l'aleatorietat de quina direcció escull l'agent per tal de sempre tenir tendència a anar cap a la meta.

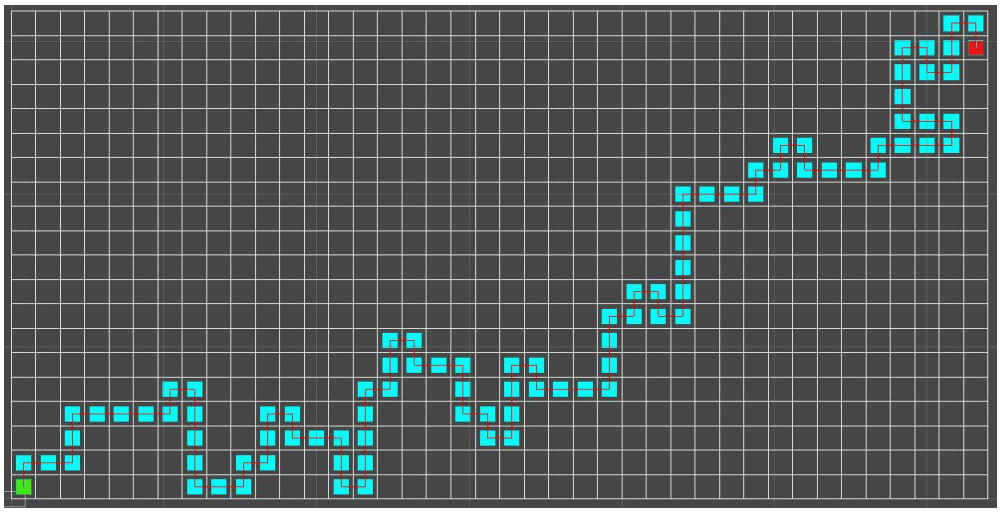

Figura 13. Exemple de Random Walk amb meta. Font: Elaboració pròpia.

Si es volgués utilitzar aquest algorisme en el treball, podríem designar diferents agents per cada una de les zones del mapa amb una meta especificada. Indicant una meta es pot aconseguir que els agents no es molestin entre ells, a més, totes les zones estarien connectades entre elles.

L'últim punt, de la mateixa forma que en l'algorisme anterior, seria generar o col·locar sales ja creades a les caselles visitades pels agents.

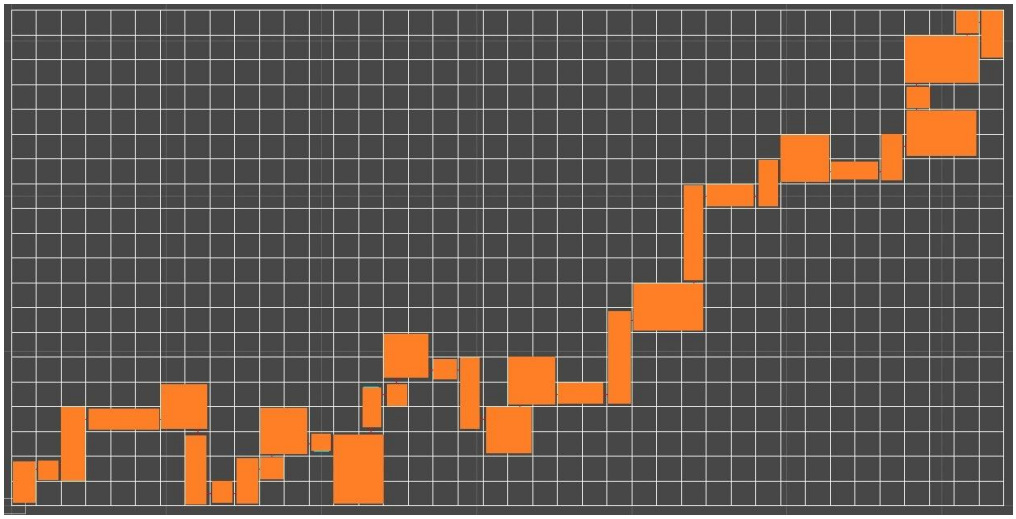

Figura 14. Exemple d'habitacions col·locades seguint la ruta d'un agent. Font: Elaboració pròpia.

4.5.3. Diffusion Limited Aggregation

Paul Bourke descriu el mètode de Diffusion Limited Aggregation com un model de processos de creixement fractal o com una rèplica de la difusió i l'agregació d'ions de zinc en una solució electrolítica sobre elèctrodes. Alguns exemples que podem trobar a la vida real que es comporten com aquest mètode són: el moviment de l'electricitat, el gebre que es forma a la finestra o les connexions que forma un riu al llarg del seu camí. (Bourke, 2006).

Durant el 1981, T.A. Witten, Jr. i L. M. Sander van desenvolupar un model de DLA utilitzant una variant del model d'Eden. El funcionament d'aquest model comença per col·locar una partícula al centre de la quadrícula, seguidament una segona partícula és afegida a una posició aleatòria allunyada del punt central, aquesta partícula es mourà de manera aleatòria fins que visiti una posició adjacent al punt central. En aquest moment, la segona partícula passa a ser part del clúster central i es col·loca una tercera partícula que es mourà de forma aleatòria fins que s'enganxi al clúster. Aquest patró es repetirà tantes vegades com partícules que es vulguin utilitzar (Witten & Sander, 1981).

En el cas que es vulgui utilitzar aquest mètode en el treball, el procés de l'algorisme seria el següent:

1. **Generar un node central:** Generar un node central a la graella a on els *walkers* o les partícules es puguin enganxar.

2. **Spawn de random walkers:** S'inicialitzaran tants *walkers* com es necessitin a tot el perímetre de la graella.

3. **Comprovació d'acabament:** Esperar fins que tots els *walkers* estiguin enganxats al clúster central.

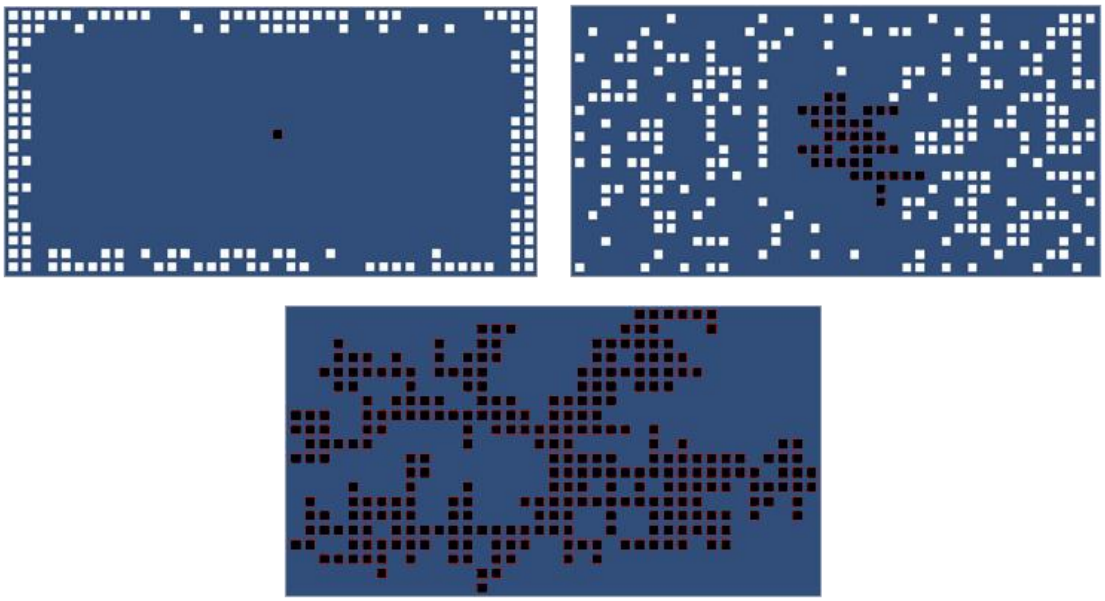

Figura 15. Exemple de Diffusion Limited Aggregation. Font: Elaboració pròpia.

Aquest mètode proporciona una estructura de xarxa a on es té absolut control de la quantitat de partícules que es volen instanciar i ens assegura tots els nodes són accessibles i estan connectats.

4.5.4. K-means (Clustering)

K-means és un algorisme de clustering, la seva funció és inicialitzar un número de centroids (anomenats k) que delimitaran els diferents grups generats per l'algorisme al final de la seva execució. Aquests centroids són col·locats de forma aleatòria en el gràfic, o en el cas d'aquest treball, a la quadrícula. Un cop col·locats, els punts o nodes més propers a un determinat centroid passaran a ser part d'un mateix grup o cluster, de tal forma que cada node queda assignat a un grup delimitat

pel centroid més proper que tingui. Finalment, s'ajusta la posició dels centroids per quedar en el centre del grup que ha format de nodes. Es repetirà l'assignació de clusters per distància als centroids i la recol·locació d'aquests un nombre x de vegades. (Na et al., 2010)

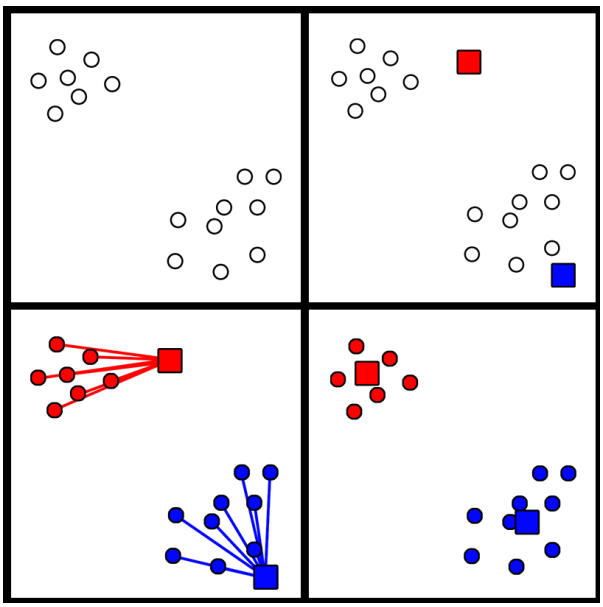

Figura 16. Exemple de clustering amb K-Means. Font: Elaboració pròpia.

El procés de l'algorisme és el següent:

1. **Generació de centroids:** Es generaran un número designat de *centroids*.

2. **Col·locació dels centroids:** Aquests seran col·locats a la graella a una posició aleatòria.

3. **Assignació de grups:** Cada node o *walker* de la graella serà assignat a un grup determinat pel *centroid* que tingui més a prop.

4. **Recol·locació dels centroids:** S'ajustarà la posició dels *centroids* per situar-se al mig de cada grup al qual formen part.

5. **Ajustament de grups:** Es repetirà el pas 3 i 4 un número x de vegades fins que el resultat sigui estable.

Aquest mètode pot ser útil per delimitar les diferents zones que formaran el mapa.

4.5.5. Binary Space Partition

La partició d'espais dintre de la generació procedimental és bastant comuna, és una eina versàtil que serveix per delimitar àrees o zones d'un espai concret. Bucklew (2017). explica que el mètode de Binary Space Partition (BSP) consisteix a dividir una àrea amb una estructura arbòria, un espai és dividit en dues subzones, aquestes dues subzones es tornaran a dividir en dos, i així constantment fins que s'arribi a un llindar específic.

Dins de la generació procedimental aquesta tècnica normalment s'utilitza per generar una habitació dintre de cada segment un cop l'espai ha estat dividit el nombre de cops desitjat. Finalment, es crearan passadissos que connectaran totes les habitacions.

El videojoc Caves of Qud per exemple, va fer ús d'aquest algorisme per la creació de les ruïnes del joc. (Bucklew, 2017, p.293).

El procés de l'algorisme és el següent:

1. **Partició vertical:** Es parteix un espai en dos utilitzant una línia vertical a un lloc aleatori.

2. **Partició horitzontal:** De les dues noves zones es tornen a partir en dos, aquest cop fent servir una línia horitzontal en un punt aleatori de la subzona en qüestió.

3. **Repetició**: Es repeteix el punt 1 i 2 fins que s'aconsegueixi arribar al número de particions desitjades.

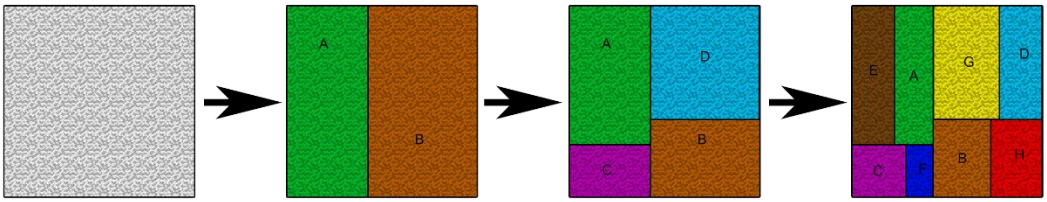

Figura 17. Exemple de Binray Space Partition. Font: Elaboració pròpia.

Aquest algorisme es pot utilitzar per delimitar les zones del mapa o per genera les habitacions de cada zona.

4.5.6. Flood Fill

Flood Fill és un algorisme recursiu a on donat uns valors numèrics o uns colors i una posició, pintarà o modificarà els valors de les caselles de la quadrícula que s'ha passat per paràmetre. Es fa servir quan es vol omplir o pintar un objecte que té una vora que defineix el límit del objecte. (Kumar et al., 2019)

Per exemple si es té una graella de 9x9 amb una línia negra que separa la quadrícula en dos [Figura 18] i es vol pintar una de les seccions d'aquesta graella, s'utilitzaria l'algorisme Flood Fill. Com a paràmetres es passaran el color original (color blanc), el color nou (color vermell) i la primera casella que es vol pintar. El primer pas serà mirar si la casella passada per paràmetre és del color original, si ho és, es pintarà del nou color vermell. Aquesta operació es repetirà per totes les caselles adjacents que siguin del color original, en aquest cas, color blanc. El resultat final serà una part de la graella pintada amb el nou color vermell, la línia negre que delimita la vora i una part sense pintar amb el color original.

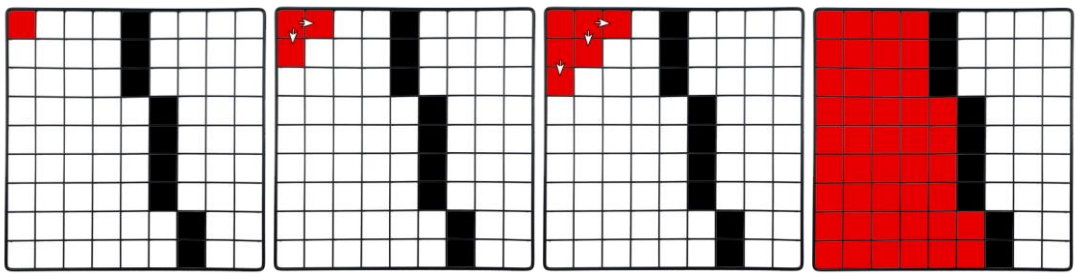

Figura 18. Exemple de Flood Fill. Font: Elaboració pròpia.

Així doncs, el procés de l'algorisme és el següent:

1. **Indicar paràmetres**: S'indicarà el valor numèric o color original, el nou valor numèric o color nou i una posició a la graella.

2. **Modificar el valor o color**: Si la casella passada per paràmetre és del valor o color original aquesta serà modificada i passarà a tenir el valor o color nou.

3. **Repetició per les caselles adjacents**: Es repetirà el segon pas per cada una de les caselles adjacents del mateix color o valor fins que no quedi cap casella amb el color o valor original.

En el cas d'aquest projecte, aquest algorisme serà útil per un cop delimitades les zones, verificar i ajustar totes aquelles caselles o nodes que haurien de ser d'una mateixa zona però que no ho són.

4.6. Minimum Spanning Tree (MST)

El Minimum Spanning Tree és una de les tècniques més típiques i conegudes gràcies a les diverses solucions que té i a la importància que ha tingut com a rol central a l'hora de dissenyar diversos algorismes de computació. (Graham & Hell, 1985)

Donat un gràfic a on tots els nodes estan connectats i cada node té un cost assignat, el MST serveix per obtenir l'estructura arbòria o el camí d'aquest gràfic amb el mínim cost total, sense cap bucle i que ens asseguri que tots els nodes formen part d'aquest camí.

Prim (1957) menciona dos principis fonamentals per tal de construir la xarxa de connexions més curta:

- Principi 1: Qualsevol node (*terminal*) aïllat pot ser connectat al veí més proper.

- Principi 2: Qualsevol grup de nodes connectats (*fragment*) aïllat pot ser connectat al veí més proper per l'enllaç disponible més curt.

Fent servir el principi 1 (P1) només un cop per produir el primer grup de nodes connectat i després aplicant successivament el principi 2 (P2) es pot aconseguir generar la xarxa de nodes amb el cost més baix a on tots els nodes són accessibles.

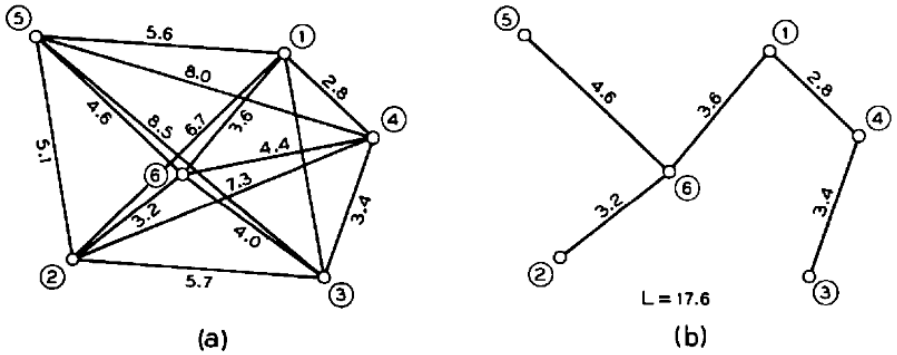

Figura 19. Exemple de gràfic de nodes utilitzant l'algorisme de Prim. Font: Prim, 1957.

Prim afegeix que fent ús del principi 1 i 2 també es pot arribar a construir un LST (Longest Spanning Tree) si es modifica el símbol de la llargada que es vol agafar, és a dir, en comptes de buscar el veí més proper d'un grup de nodes connectats es busca el veí més llunyà. (Prim, 1957)

El procés de l'algorisme és el següent:

1. **Generació de llistes**: Es generaran dues llistes, una servirà per registrar tots els nodes visitats que conformen un grup (*fragment*) i l'altre per tenir un registre de tots els nodes aïllats no visitats (*isolated terminals*).

2. **Comparació de distàncies**: Es compararan totes les distàncies dels nodes aïllats amb els nodes de grup ja visitats. S'afegirà a la llista de nodes visitats el node aïllat més proper al grup amb una referència del node pare i seguidament s'eliminarà de la llista de nodes no visitats.

3. **Repetició**: Es repetirà el segon pas fins que tots els nodes formin part de la llista de nodes visitats.

En el cas d'aquest projecte, ens servirà per generar les rutes entre habitacions de cada zona i l'ordre o ruta de zones.

4.7. Editor personalitzat de Unity

El motor de Unity permet crear o editar el seu editor per personalitzar l'eina al gust dels desenvolupadors, de tal manera que puguin treballar més fàcilment o de manera òptima. Hi ha molts exemples d'eines creades per usuaris que es poden obtenir a la botiga digital que té Unity, com per exemple una eina de pathfinding en A* creada per Aron Granberg o una eina que facilita la creació de jocs rítmics desenvolupada per Benny Kok.

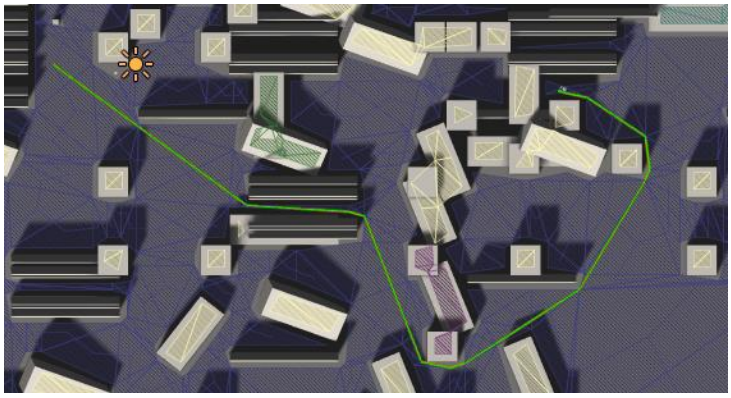

Figura 20. Exemple d'eina personalitzada de Unity desenvolupada per Aron Granberg. Font: Unity Asset Store, s.d.

Benny Kok és l'autor també d'un llibre titulat Beginning Unity Editor Scripting a on ensenya a modificar l'editor per adaptar-lo a les necessitats de cada un. Kok explica que hi ha diversos mètodes per crear editors personalitats a Unity i remarca la importància de crear eines amigables per l'usuari que les farà servir, explica com afegir descripcions, textos d'ajuda, fer botons o dibuixar referències visuals a la pantalla (Kok, 2021).

Les eines o editors personalitzats de Unity també són útils per ampliar el repertori d'opcions que pot modificar l'usuari i d'aquesta manera aconseguir un resultat més especialitzat.

5. Disseny metodològic i cronograma

En aquest apartat s'explica el model metodològic seguit pel desenvolupament d'aquest projecte i el cronograma amb una previsió del compliment de cada fita.

5.1. Disseny metodològic

Per tal de dur a terme aquest projecte, es farà servir el model en espiral de Barry Boehm (Boehm, 1988). Aquest model adaptatiu està dissenyat per gestionar els riscs del projecte en el qual s'aplica i poder reaccionar per tal de mitigar-los. Un dels factors característics d'aquest model és que es representa en espiral en comptes d'una seqüència o llista com altres tipus de metodologies de treball.

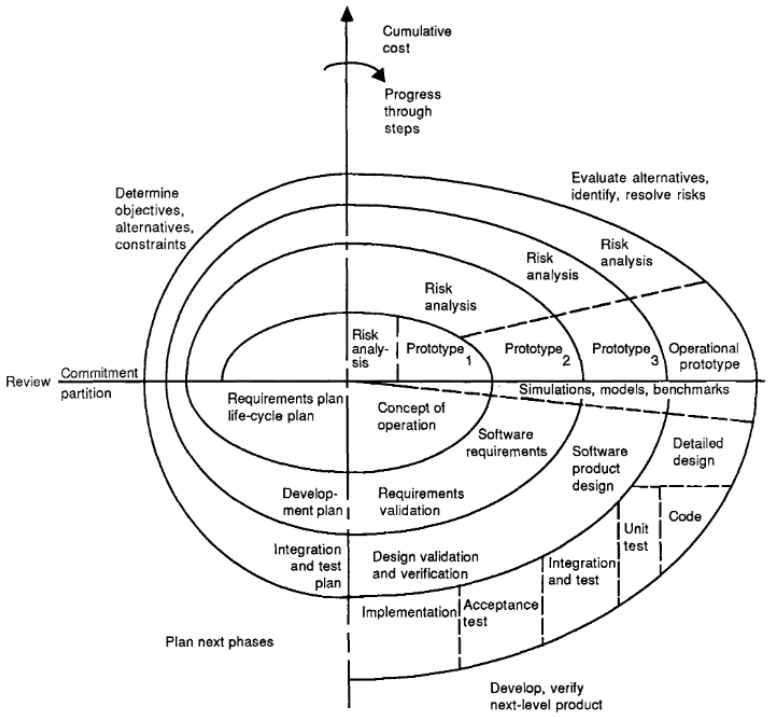

Figura 21. Il·lustració del model espiral. Font: Boehm, 1988.

A cada una de les iteracions de l'espiral està dividida en quatre seccions:

1. **Determinar els objectius, alternatives i limitacions:** En aquesta primera part, tracta d'especificar els objectius segons la fase en la qual

ens trobem del projecte, avaluar les limitacions i planificació d'alternatives per complir els objectius establerts.

2. **Avaluar les alternatives, identificar i solució de riscs:** En aquesta segona part es tracta d'investigar i avaluar els riscs identificats per tal de reduir-los o eliminar-los. Per aquest motiu, és possible que es necessitin fer prototips o avaluacions comparatives en aquest apartat.

3. **Desenvolupar i verificar:** Un cop avaluats els riscs i les alternatives es decideix quin camí seguir per tal de desenvolupar els objectius establerts. També es verifica el funcionament i l'estabilitat del programari creat.

4. **Planificació:** Finalment, aquest últim punt, es fa una revisió de tot el programari desenvolupat al llarg d'aquest cicle i es decideix si passar a la següent iteració de l'espiral o no. En cas de continuar, es planifica la següent iteració.

5.2. Planificació d'iteracions del projecte

Així doncs, aquest projecte tindrà una espiral de 7 cicles, seguidament es detallarà cada un d'aquestes iteracions:

5.2.1. Iteració 1: Investigació

La primera iteració se centra a recollir la informació necessària per a dur a terme el projecte. Acaba amb l'entrega de l'avantprojecte.

5.2.2. Iteració 2: Generació de mapa

En aquesta iteració, l'objectiu principal és generar l'àrea que delimitarà el mapa amb els algorismes estudiats a la prèvia iteració de l'espiral.

5.2.3. Iteració 3: Delimitació de zones

La tercera iteració se centra a delimitar les diferents zones en el mapa generat en el cicle previ i corregir els nodes que no estiguin assignats correctament.

5.2.4. Iteració 4: Generació d'habitacions

La finalitat d'aquesta iteració és generar les habitacions interconnectades de cada zona.

5.2.5. Iteració 5: Generació de rutes

L'objectiu d'aquesta iteració és generar la ruta de cada zona i l'ordre en el qual es travessarà el mapa.

5.2.6. Iteració 6: Assignació de rols a les sales creades

En aquesta fase es generaran les habitacions especials com les sales de transició, sales de guardar partida, sales de teleportació o sales de *boss*.

5.2.7. Iteració 7: Editors personalitzats

En aquesta fase l'objectiu serà desenvolupar les finestres o editors personalitzats per així tenir una eina més usable i personalitzable.

5.3. Cronograma

Iteracions	Desembre	Gener	Febrer	Març	Abril	Maig	Juny
Desenvolupament de la memòria							
Iteració 1							
Iteració 2							
Iteració 3							
Iteració 4							
Iteració 5							
Iteració 6							
Iteració 7							

Taula 2. Taula del cronograma. Font: Elaboració pròpia.

6. Desenvolupament i resultats

En aquest apartat s'expliquen les decisions i els procediments utilitzats durant les diverses fases o iteracions del projecte. Es detallen els problemes i les solucions trobades així com els resultats de cada fase.

6.1. Preparació

Tal com s'ha mencionat anteriorment, s'ha escollit Unity com a motor de desenvolupament del projecte, concretament es farà servir la versió 2020.3f1.

Per poder escriure tot el codi s'utilitzarà Microsoft Visual Studio Community 2022 com a IDE principal i el projecte serà codificat en C#.

6.2. Primera fase

Els objectius de la primera fase eren fer una primera investigació per tal d'adquirir els coneixements necessaris per dur a terme el projecte, començar a fer petites *demos* amb la finalitat de posar-los a prova i la creació de la primera base a on es construiria tot el mapa.

6.2.1. Creació de la quadrícula

Per tal de generar el mapa es va optar per fer una quadrícula com a base. En tractar-se d'un mapa continuo sense salts o espais entre zones va fer que fos més fàcil col·locar i controlar les sales amb exactitud, a més, d'aquesta manera es té l'opció a ajustar les dimensions del mapa a l'instant de manera dinàmica.

Així doncs, es va crear una classe anomenada Grid que s'encarrega de generar la quadrícula. Per tal de fer la graella es necessiten quatre dades:

- **Amplada:** Amplada de cada cel·la de la quadrícula.
- **Alçada:** Alçada de cada cel·la de la quadrícula.
- **Número de columnes:** Nombre de columnes que tindrà la quadrícula.
- **Número de files:** Nombre de files que tindrà la quadrícula.

A part de la classe Grid, es va crear una classe anomenada GridGenerator amb la finalitat de passar les dades de l'inspector de Unity a la classe Grid utilitzant el seu constructor. A més, el GridGenerator és l'encarregat de pintar la quadrícula a la pestanya de Scene de Unity.

6.2.2. Diagrama de classes

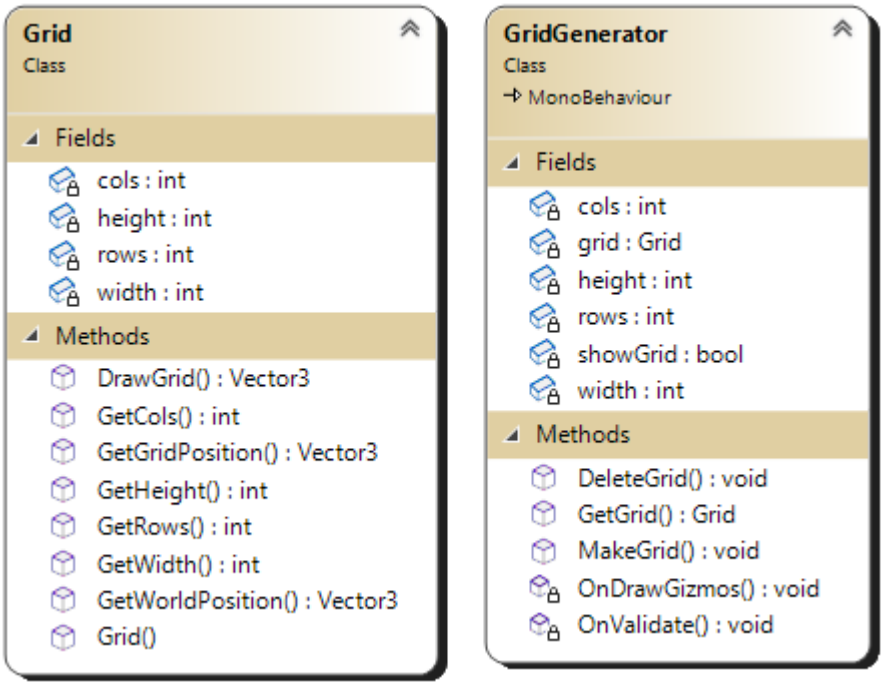

Figura 22. Diagrama de les classes Grid i GridGenerator. Font: Elaboració pròpia.

6.3. Segona fase

En aquesta segona fase l'objectiu principal era generar l'estructura del mapa dins de la quadrícula creada a la fase anterior, dit d'una altra manera, determinar la forma i les posicions que formaran part del mapa.

Per tal de fer-ho es va decidir utilitzar el mètode de DLA (Diffusion Limited Aggregation), ja que és fàcil i ràpid d'implementar, genera un mapa que sempre estarà interconnectat i es pot escollir la quantitat de les cèl·lules o walkers que s'utilitzaran en el procés.

Tot i que al final es va decidir fer ús del mètode DLA, cal remarcar que no és el més òptim. El mètode RRP (Random Room Placement), per exemple, seria més ideal en el cas que aquest projecte fos a sortir al mercat, ja que es té molt més control de les habitacions i de quin tipus de contingut contenen. A més, es podria arribar a implementar un sistema de plantilles per poder ajustar ràpidament les sales o zones a les nostres necessitats. A causa del temps limitat i en tractar-se d'un projecte universitari, es va decidir fer ús del mètode DLA, ja que és més genèric i més visual a l'hora de representar com es genera el mapa.

El principal problema de fer servir DLA, però, resideix en el poc control que es té sobre els walkers. Tal com s'ha mencionat anteriorment, l'única funció dels walkers és moure's aleatòriament per la quadrícula fins que topin amb el nucli central o un altre walker enganxat a aquest. Per aquest motiu, els temps de generació estan proporcionalment lligats amb el comportament aleatori dels walkers i amb determinades situacions, com per exemple, en cas d'haver-hi poca quantitat de walkers envers la mida del mapa. Per això els temps de generació poden ser molt llargs o fins i tot infinits, si es donés el cas que els walkers mai topen contra el nucli central o altres walkers enganxats a aquest.

Una possible solució a aquest problema seria limitar l'aleatorietat dels moviments que fan els walkers de tal manera que sempre estiguin una mica atrets cap al punt central de la quadrícula o directament indicar quines posicions tenen disponibles quan es generen els walkers, de tal manera que quedin enganxats directament només aparèixer. En el següent exemple [Figura 23], es pot veure els quadrats vermells com aquells walkers que ja estan enganxats i els quadrats verds com les possibles posicions que tindrien disponibles els següents walkers quan es generessin.

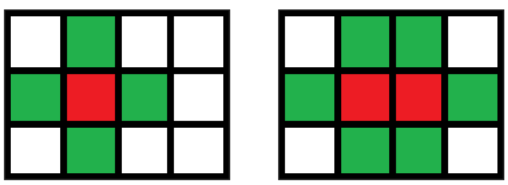

Figura 23. Exemple de la solució proposada. Font: Elaboració pròpia.

Així doncs, per poder implementar el mètode DLA es van crear les classes Walker i MapGenerator.

6.3.1. Classe Walker

La classe Walker és l'encarregada de la lògica de cada walker, gestiona el moviment per la quadrícula, control de col·lisions i canvi d'estat quan s'han d'enganxar.

Per aquest treball només es va limitar el moviment dels walkers per tal que sempre estiguin dins la quadrícula.

6.3.2. Classe MapGenerator

La funció principal de la classe MapGenerator és instanciar i col·locar els walkers dins de la quadrícula juntament amb un nucli o punt central a on es poden enganxar. El generador de mapes també serà l'encarregat de generar una llista de tots els nodes o walkers enganxats, per així tenir un registre de la posició final de cada un.

Per tal d'evitar que el mapa estigues més carregat en una àrea de la quadrícula que en una altra, es va decidir que el punt d'origen dels walkers estigues repartit per tota la vora de la quadrícula, d'aquesta manera es van evitar problemes com instanciar un walker a sobre d'un altre que ja formi part del clúster o una sobrecàrrega de walkers a una zona especifica.

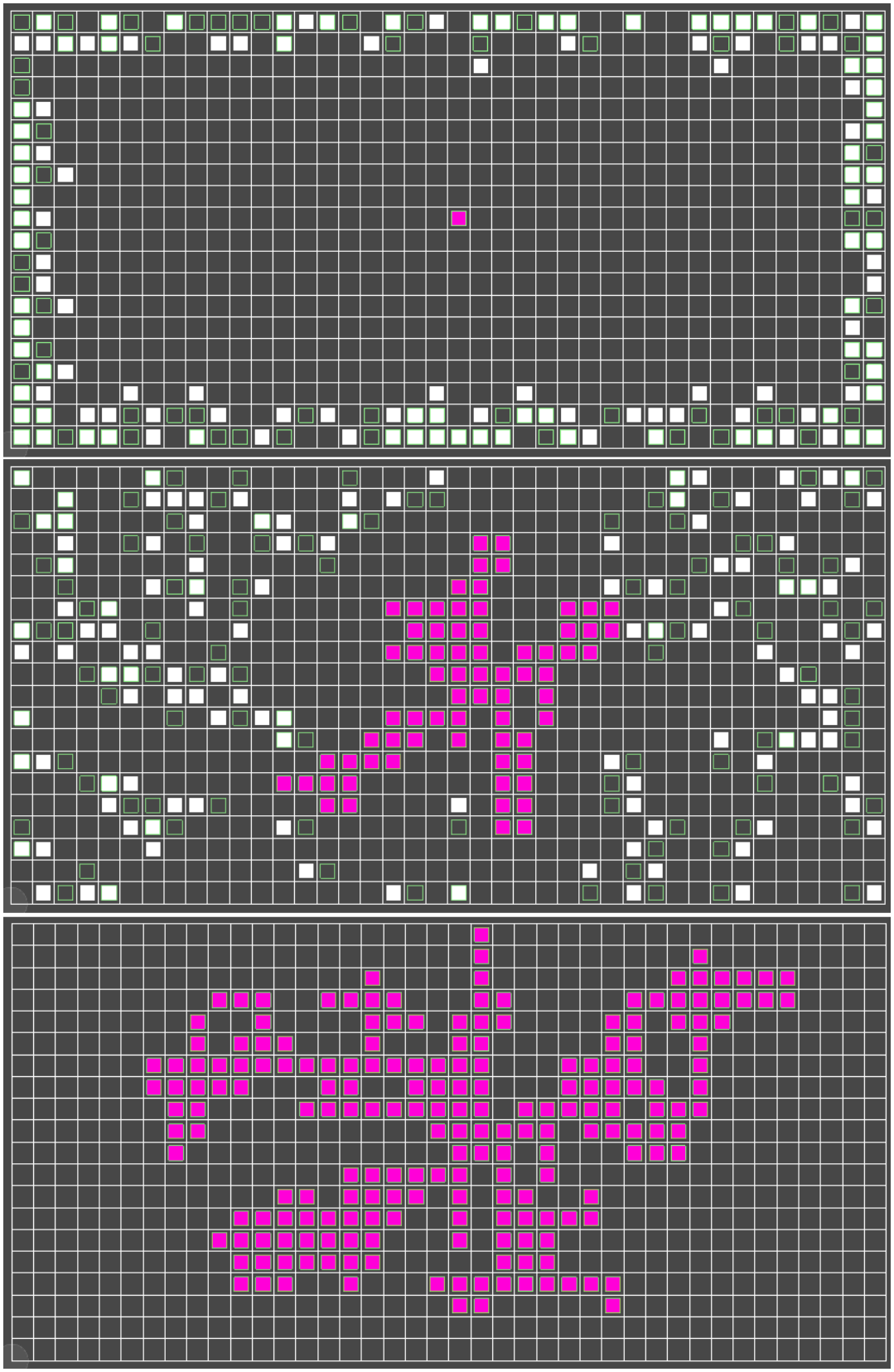

Figura 24. Exemple de generació de mapa per DLA. Font: Elaboració pròpia.

6.3.3. Diagrama de classes

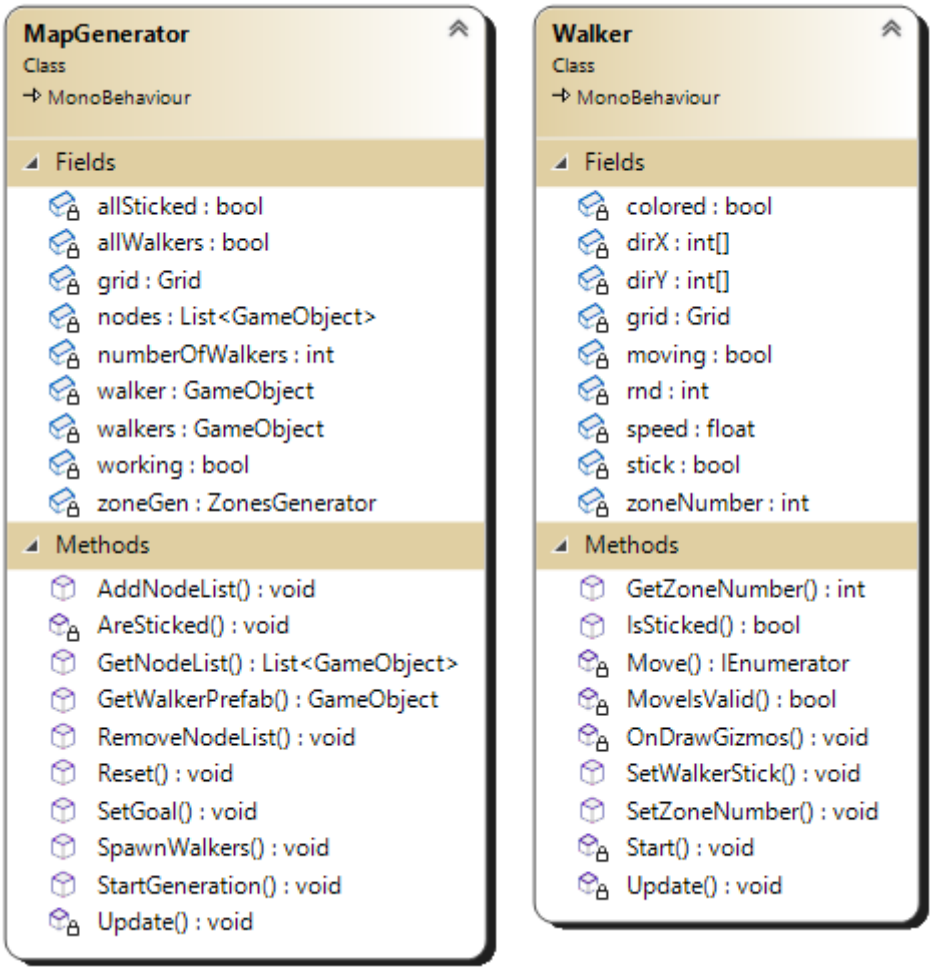

Figura 25. Diagrama de les classes Walker i MapGenerator. Font: Elaboració pròpia.

6.4. Tercera fase

Per la tercera fase, l'objectiu era delimitar les diverses zones que formen el mapa d'un joc Metroidvania. Per tal d'aconseguir-ho, es va utilitzar el mètode K-means. En comptes del K-means es podria haver fet servir el mètode de BSP (Binary Space Partition) per tal d'assolir aquesta mateixa finalitat, però les zones resultants haurien quedat massa simètriques a causa dels talls rectangulars de dividir constantment l'espai en dos. En contrast, fent ús del K-means s'obté un

resultat més net, en el sentit que no es veuen talls, i que hi ha diversitat de mides entre les diferents zones.

Tanmateix, el sistema de K-means tampoc és perfecte i es pot donar el cas que un node estigui més a prop d'un centroid que clarament no forma part del grup de nodes del que pertany, i per això, pintar-se d'un color que no hauria. En aquests casos s'utilitzarà el Flood Fill per corregir aquests errors.

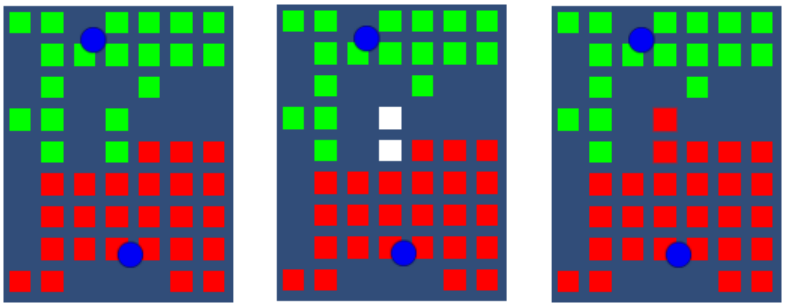

Figura 26. Exemple de Flood Fill corregint el resultat de K-Means. Font: Elaboració pròpia.

Cal remarcar que durant el desenvolupament del projecte es va decidir limitar el nombre de zones amb un mínim d'una sola zona i un màxim de sis zones.

A causa d'aquesta limitació es van generar diversos camps que tenen a veure amb les zones de manera manual, com per exemple un array de colors amb una mida fixa de sis colors o sis llistes de nodes diferents per cada una de les possibles zones. Idealment, totes aquestes llistes haurien de ser dinàmiques, de tal forma que només es construïssin tantes llistes com zones delimitades i permetessin tenir un rang de zones molt més ampli, però per raons de *debug*/*testing* i en tractar-se d'un projecte no comercial es va decidir fer els camps manualment.

Així doncs, es va fer un script anomenat ZoneGenerator per tal de complir l'objectiu de la tercera fase.

6.4.1. Classe ZoneGenerator

La classe ZoneGenerator s'encarrega d'executar principalment cinc passos per tal de generar les zones:

6.4.2. ZoneGenerator – Generació de centroids

El primer pas per tal d'aconseguir dividir el mapa en zones, és obtenir la llista de walkers que s'ha obtingut en el generador de mapes i col·locar tants centroids a llocs aleatoris de la quadrícula com zones delimitades prèviament a l'inspector de Unity. En el projecte els centroids s'han anomenat *pointers*.

6.4.3. ZoneGenerator – Ajustament de centroids

Tot seguit, s'assignarà cada walker al centroid que tinguin més a prop. Un cop tots els walkers estiguin assignats a un centroid, s'ajustarà la posició dels centroids perquè quedi al centre del seu grup de walkers. Es repetirà aquesta operació fins que la posició dels centroids sigui estable.

Per tal de facilitar la visualització dels grups de walkers creats pels diferents centroids es va decidir pintar-los d'un color diferent:

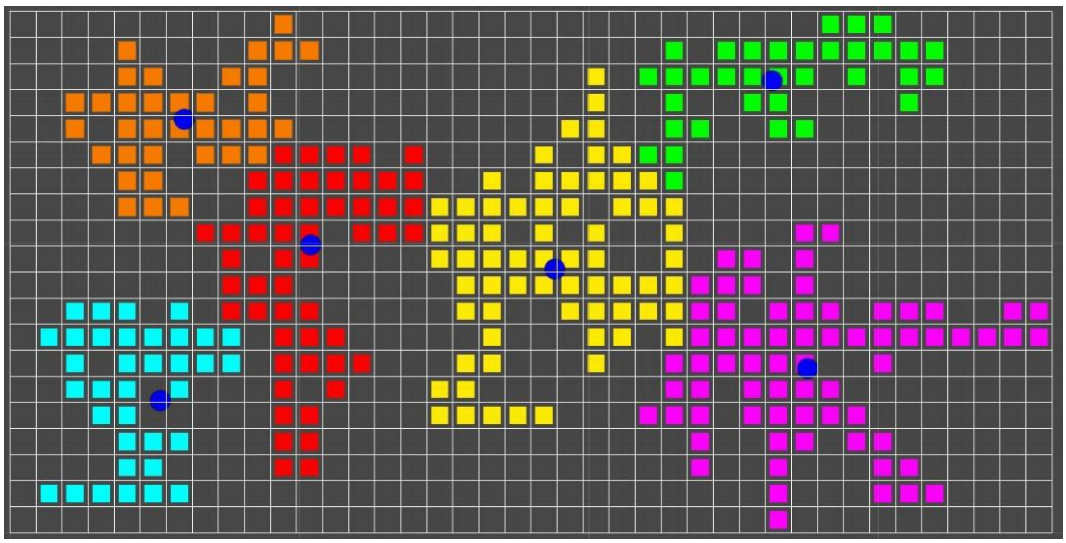

Figura 27. Exemple de mapa amb zones fent servir K-means. Font: Elaboració pròpia.

6.4.4. ZoneGenerator – Omplir les llistes de zones

Un cop tots els walkers formen part d'un grup, se'ls col·locarà a una llista de zona. Aquestes llistes serveixen per tenir un registre de cada grup i dels walkers que en formen part.

6.4.5. ZoneGenerator – Reassignació dels walkers incorrectes

Seguidament, es reassignaran tots els walkers que han sigut col·locats a un grup equivocat utilitzant el mètode Flood Fill tal com s'ha mencionat anteriorment.

6.4.6. ZoneGenerator – Amagar els centroids i els walkers

Finalment, s'amagaran tots els centroids i els walkers de les finestres de Unity, ja que a partir d'aquest punt ja han complert la funció principal i no són necessaris a la vista de l'usuari.

6.4.7. Diagrama de classes

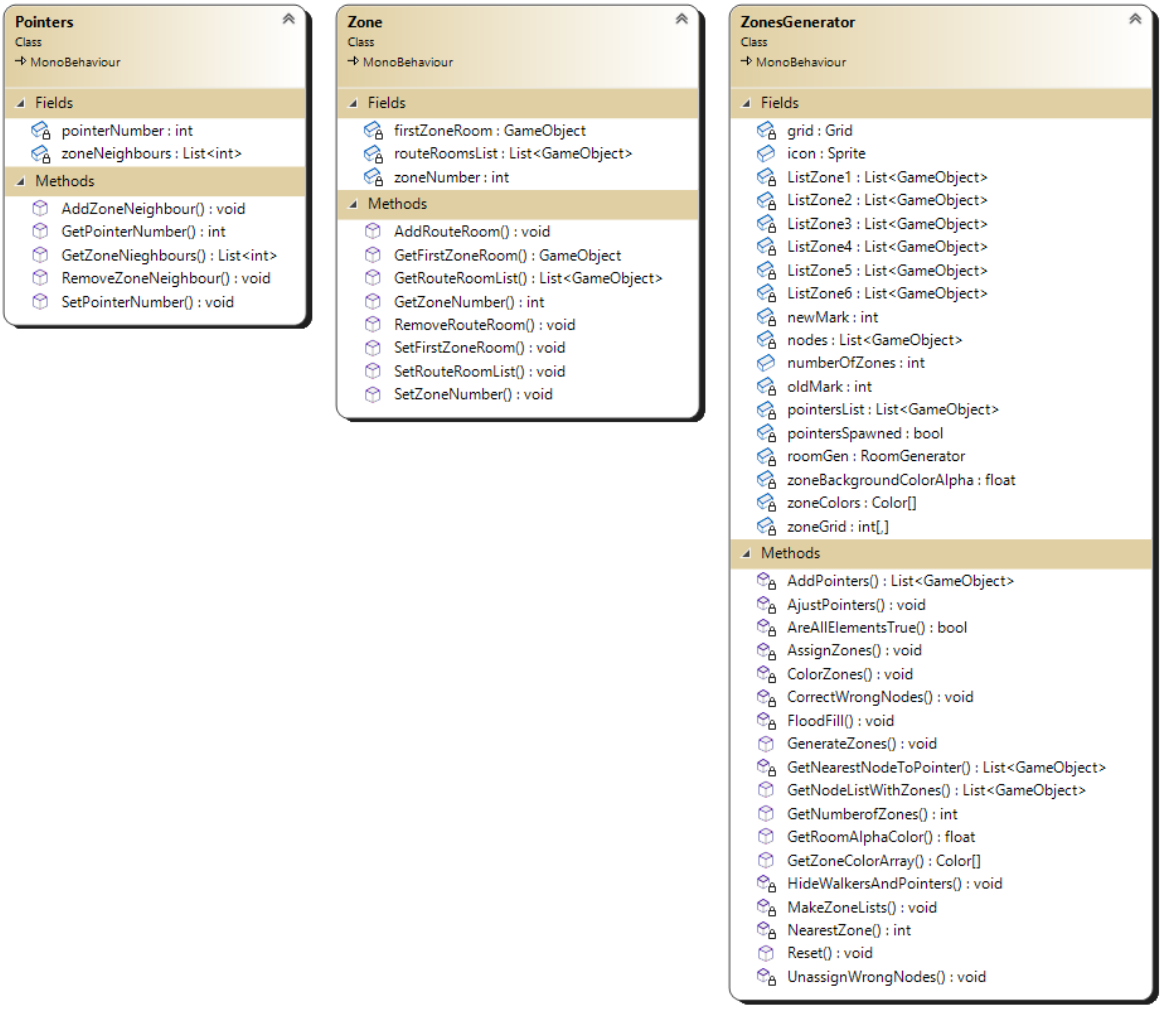

Figura 28. Diagrama de la classes ZonesGenerator, Zone i Pointers. Font: Elaboració pròpia.

6.5. Quarta fase

L'objectiu de la quarta fase va ser generar les habitacions del mapa utilitzant les llistes de zones generades a la fase anterior.

6.5.1. Habitació modular

Per tal d'aconseguir això, es va optar per crear un prefab d'una habitació modular composta de diverses parets. El comportament d'aquest prefab està controlat per un script anomenat RoomCell que és l'encarregat de fer que les parets s'activin o desactivin per adaptar-se a cada situació possible.

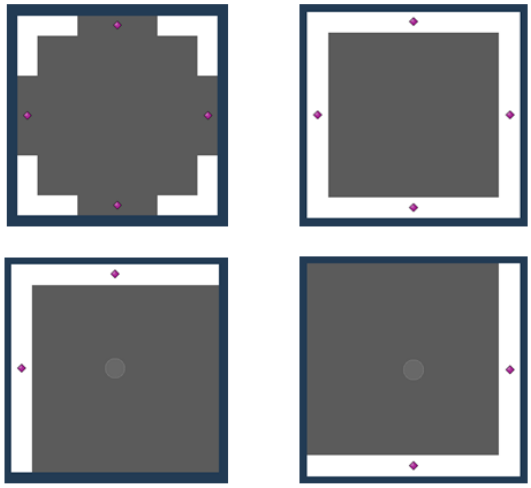

Figura 29. Habitació modular. Font: Elaboració pròpia.

A part de controlar les parets, l'script també guarda la posició, el número de la zona, el número d'habitació que té assignat, un array d'habitacions veïnes que té i diversos booleans que controlaran el tipus d'habitació i el comportament d'aquesta.

Per tal de generar totes les habitacions de cada zona es va crear un script anomenat RoomGenerator.

6.5.2. Classe RoomGenerator

Dins del RoomGenerator es generen quatre arrays nous amb la mida de la quadrícula original:

- **ArrayIntGrid**: Array numèric que assigna un número a cada habitació d'una mateixa zona.
- **ArrayColHeight**: Array numèric que comunica l'altura de cada columna de walkers.
- **ArrayZoneNumber**: Array numèric que comunica el número de la zona en la qual pertany una habitació.
- **ArrayBoolGrid**: Array de booleans d'una zona especifica que serveix per saber si una casella està ocupada o no.

Per la generació d'habitacions s'ha dividit el procés en cinc passos que es repetiran tants cops com el nombre de zones delimitades:

6.5.3. Generació d'habitacions – Omplir l'ArrayBoolGrid

El primer pas consisteix a omplir l'ArrayBoolGrid amb les posicions dels walkers que formen part d'aquella zona. El resultat d'aquest primer pas és un array bidimensional de booleans que indica si una posició està ocupada o no. Com que aquest pas es repeteix a cada iteració es té el benefici de tenir tots els altres walkers d'altres zones emmascarats.

6.5.4. Generació d'habitacions – Omplir l'ArrayColHeight

El segon pas consisteix a omplir l'array d'ArrayColHeight utilitzant l'ArrayBoolGrid per saber l'altura màxima de cada columna de walkers a la quadrícula.

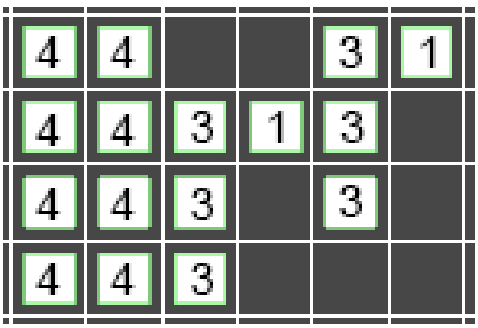

Figura 30. Exemple de l'ArrayColHeight. Font: Elaboració pròpia.

6.5.5. Generació d'habitacions – Assignació de número d'habitació

La finalitat del tercer pas és recórrer la quadrícula, primer d'esquerra a dreta i després de dreta a esquerra, i assignar un número d'habitació a l'ArrayIntGrid.

Per tal d'assignar aquest valor es comproven els veïns de cada walker utilitzant l'ArrayColHeight i l'ArrayBoolGrid. En cas que el valor d'altura de la columna veïna sigui igual o més gran que l'actual es fusionarà amb la columna actual per formar una habitació.

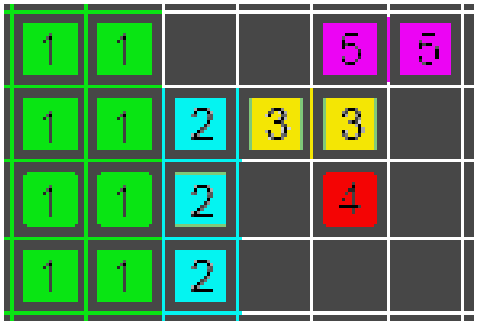

Figura 31. Exemple de l'ArrayIntGrid. Font: Elaboració pròpia.

6.5.6. Generació d'habitacions – Habitacions modulars

L'objectiu del quart pas és instanciar les habitacions modulars anomenades RoomCells a cada lloc ocupat de l'ArrayBoolGrid i actualitzar les seves parets per formar les habitacions més grans. Aquestes habitacions més grans formaran un nou gameobject anomenat Room seguit del número que s'ha assignat a aquella habitació en concret a l'ArrayIntGrid.

També s'assignarà un script a totes les habitacions anomenat Room, aquest script incorpora informació útil de l'habitació, com per exemple el seu número, la zona en què pertany, els veïns que té o les dimensions generals de l'habitació.

6.5.7. Generació d'habitacions – Organitzar les habitacions

Finalment, el cinquè pas consisteix a ordenar les habitacions i les zones a l'inspector de Unity en una estructura de pares i fills:

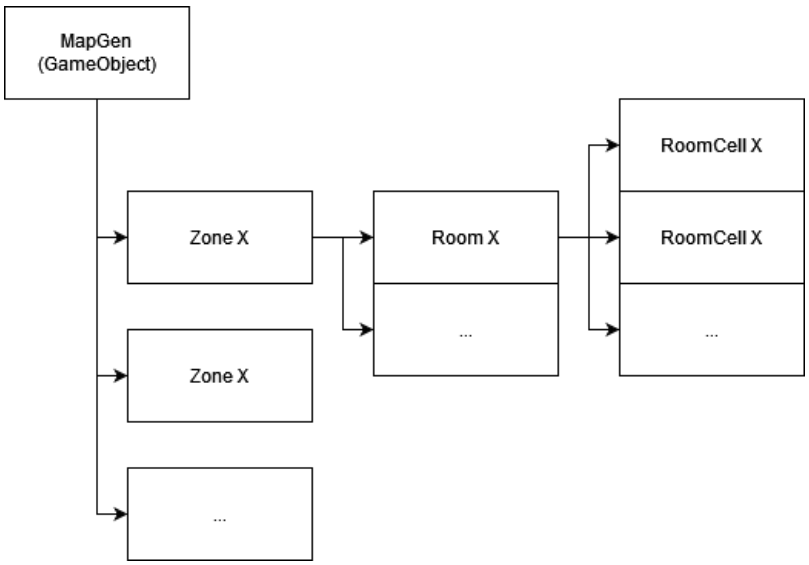

Figura 32. Jerarquia d'organització de les habitacions i les zones. Font: Elaboració pròpia.

6.5.8. Generació d'habitacions – Resultats

El resultat dels passos anteriors és un conjunt d'habitacions de diferent mida i forma separades per zones i organitzades sota una estructura de pares i fills.

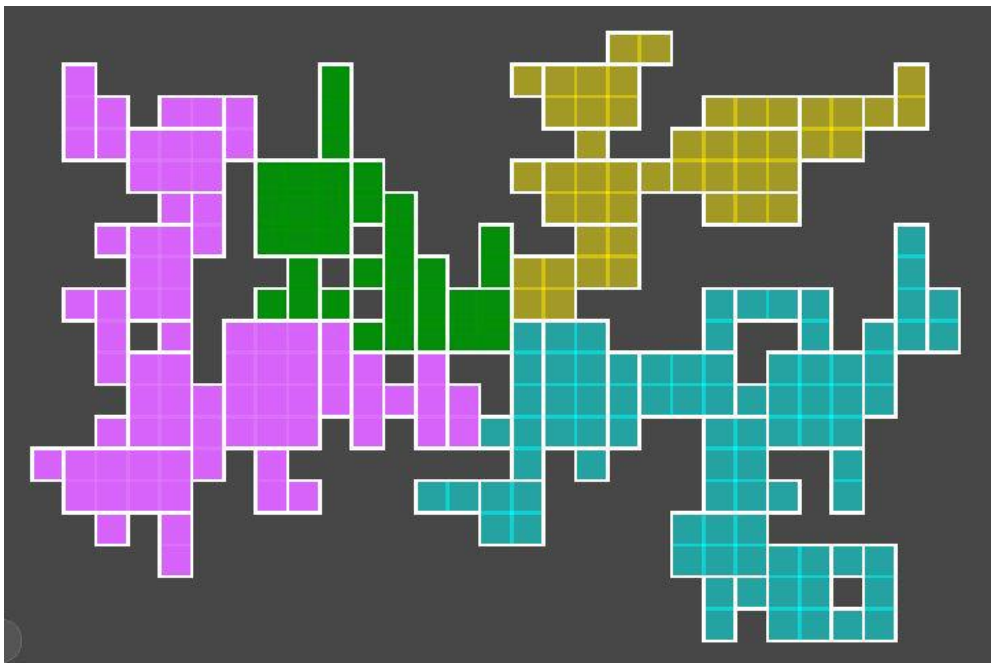

Figura 33. Exemple de zones amb les seves habitacions. Font: Elaboració pròpia.

6.5.9. Diagrama de classes

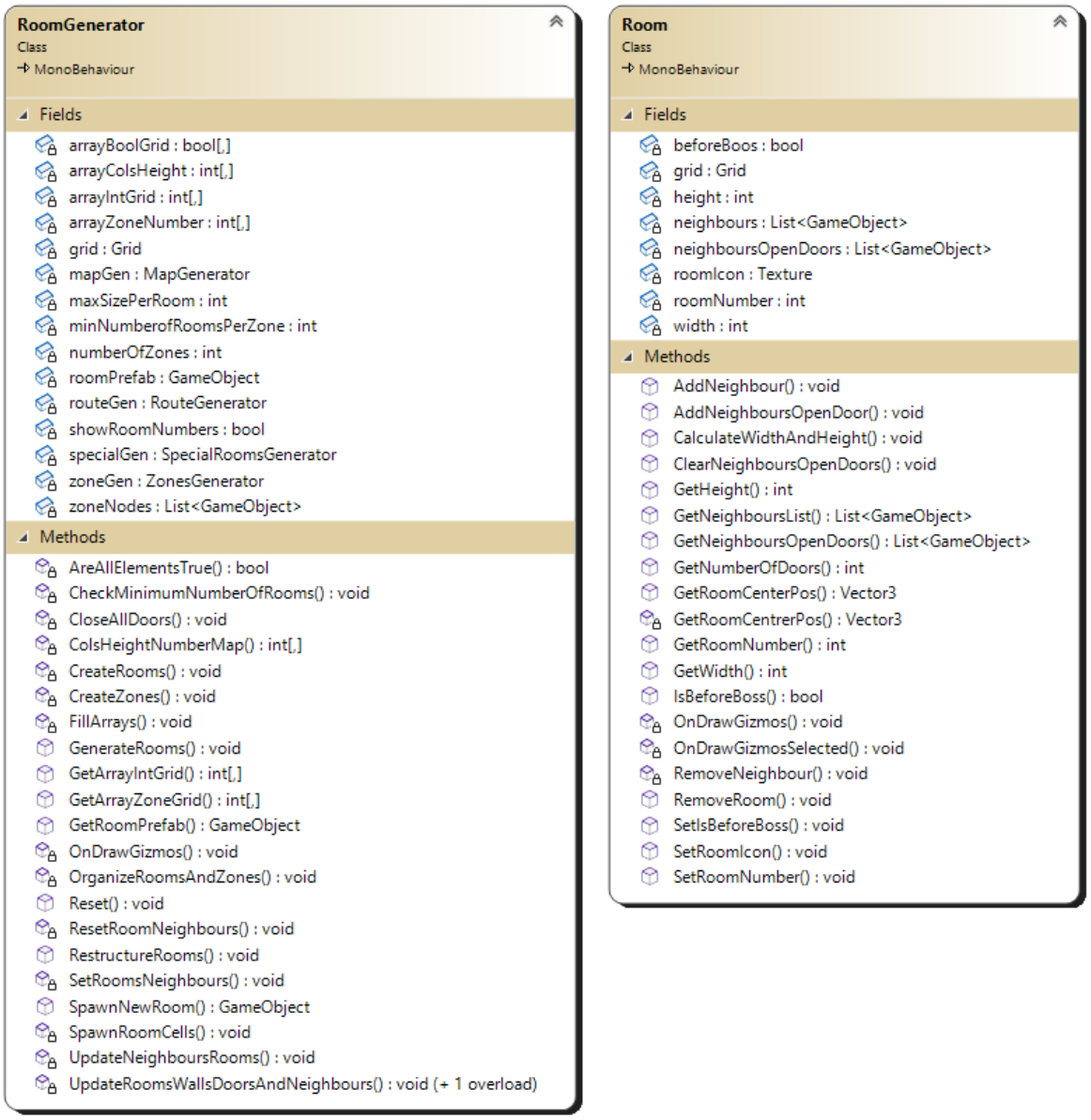

Figura 34. Diagrames de les classes Room i RoomGenerator. Font: Elaboració pròpia.

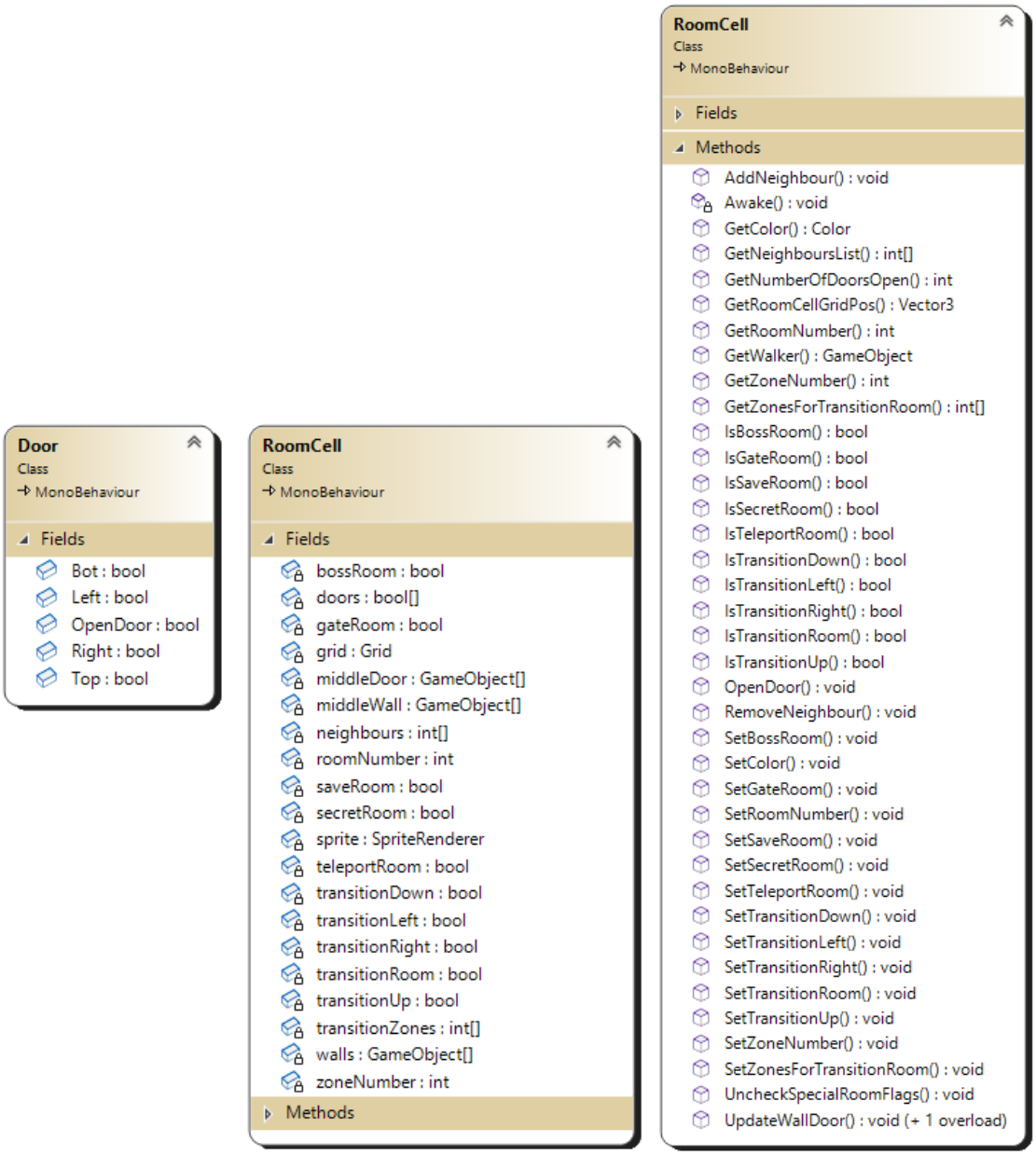

Figura 35. Diagrama de classes Door i RoomCell. Font: Elaboració pròpia.

6.6. Cinquena fase

L'objectiu de la cinquena fase era generar les rutes per on passarà el jugador. Es van generar dos tipus de ruta, la ruta d'habitacions de cada zona i la ruta d'ordre de zones.

Per aconseguir-ho es va generar un gràfic de nodes a on es va aplicar un algorisme de MST (*minimum spanning tree)* per generar un segon gràfic a on tots

els nodes eren accessibles però amb el camí més curt possible. En aquest cas es va utilitzar l'algorisme de Prim, ja que la seva implementació és ràpida i senzilla.

Així doncs, es va crear un nou script anomenat RouteGenerator per dur a terme aquesta tasca.

6.6.1. RouteGenerator – Generació del gràfic

Per tal de generar el gràfic es generen diversos nodes, cada habitació representa un d'aquests nodes. Seguidament, es dibuixen les connexions entre les habitacions. Cada connexió representa una habitació veïna que està en contacte amb aquell node o habitació especifica.

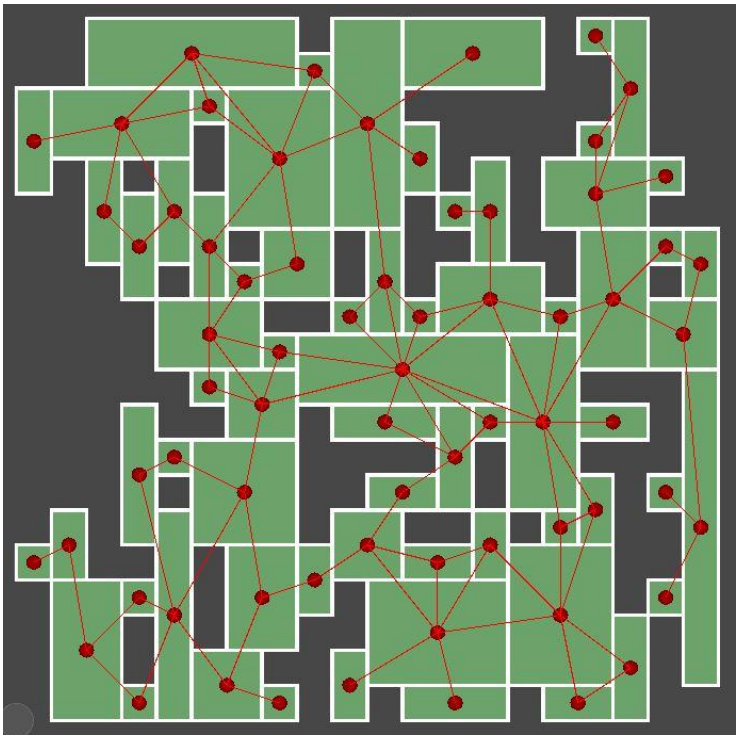

Figura 36. Exemple de gràfic de nodes generat. Font: Elaboració pròpia.

6.6.2. RouteGenerator – Generació de ruta

Seguidament, es crea una ruta que assegura l'accés a totes les habitacions utilitzant l'algorisme de Prim, ja que ens proporciona la ruta més curta d'un gràfic, però amb l'assegurança que tots els nodes són accessibles.

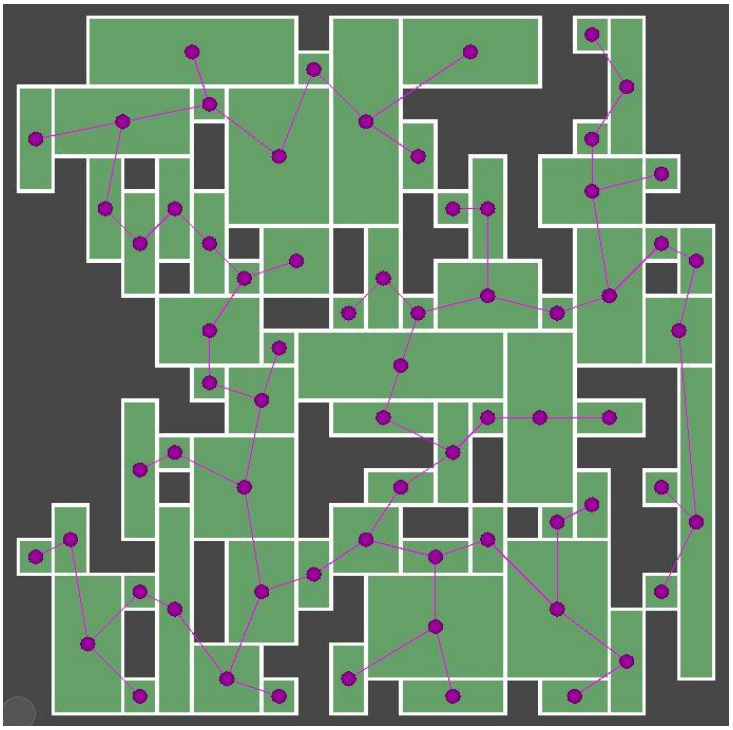

Figura 37. Exemple de ruta generada utilitzant l'algorisme de Prim. Font: Elaboració pròpia.

6.6.3. RouteGenerator – Generació de ruta de zones

Per tal de generar l'ordre de zones que el jugador haurà de travessar, es va reutilitzar el sistema de gràfic i l'algorisme de Prim, amb la principal diferencia que aquest cop, en comptes de buscar la ruta més curta entre nodes s'ha buscat la més llarga. El resultat és una llista numèrica a on els números representen les zones ordenades segons l'algorisme.

6.6.4. RouteGenerator – Obrir portes

Una vegada s'obtenen les rutes d'habitacions de cada zona s'obriran les portes de cada RoomCell per habilitar el pas segon la ruta establerta. Si hi ha més d'una RoomCell en contacte amb l'habitació que s'ha d'obrir una porta es decidirà una d'aquestes aleatòriament.

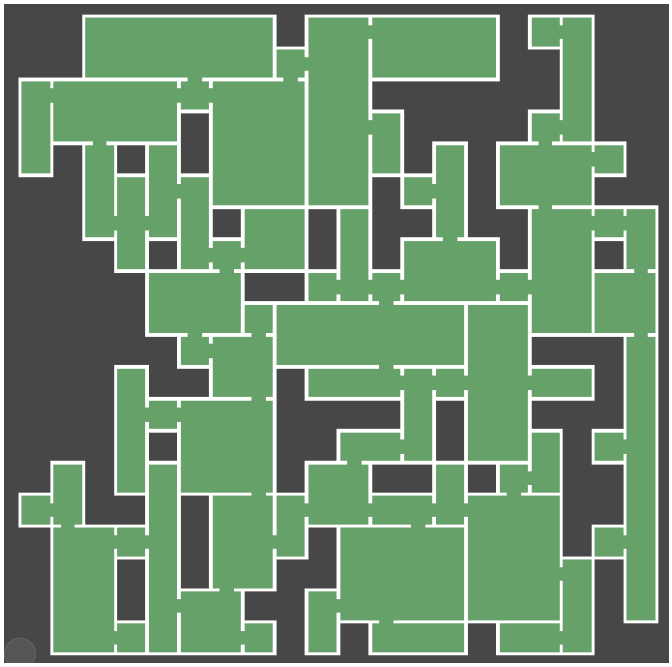

Figura 38. Exemple de mapa amb les portes obertes seguint una ruta concreta. Font: Elaboració pròpia.

6.6.5. Diagrama de classes

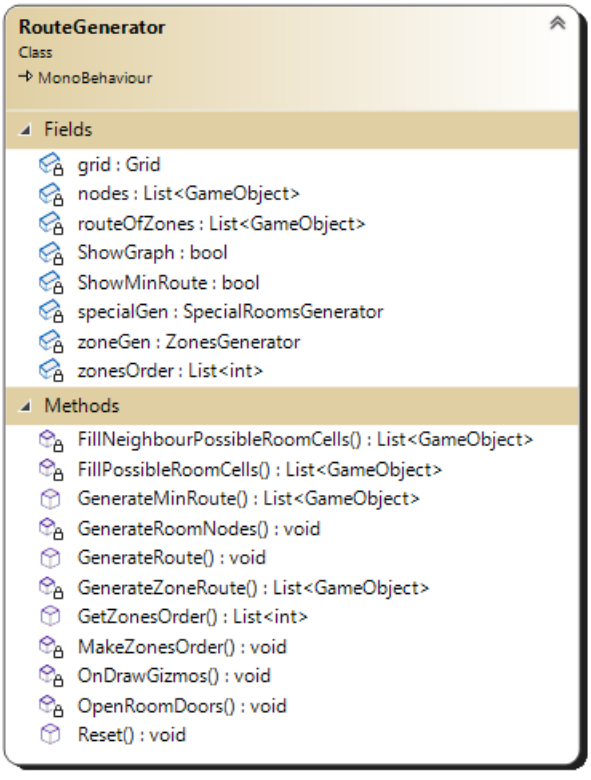

Figura 39. Diagrama de la classe RouteGenerator. Font: Elaboració pròpia.

6.7. Sisena fase

En aquesta fase l'objectiu principal era col·locar o generar les habitacions especials: habitació per guarda partida, habitació de transició, habitació de teleportació, etc. L'encarregat de dur a terme aquesta tasca serà un script anomenat SpecialRoomsGenerator.

6.7.1. Sales de transició

Les sales de transició són sales especials amb l'única funció de connectar dues zones entre elles. Per tal de generar aquestes sales es busca en el perímetre de les dues zones un espai buit a on poder col·locar l'habitació que les connecti:

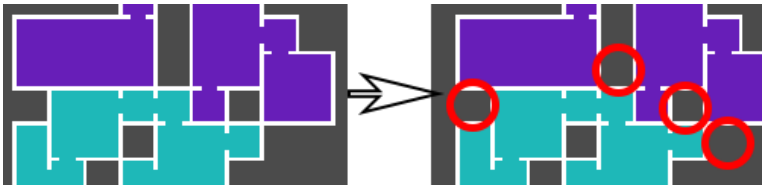

Figura 40. Exemple de posicions possibles per col·locar les sales de transició. Font: Elaboració pròpia.

En cas que les dues zones estiguin molt separades i no hi hagi cap espai buit disponible, es generarà un pont entre elles:

Figura 41. Exemple de pont generat per connectar dues zones. Font: Elaboració pròpia.

Si les dues zones no es poguessin connectar per qualsevol motiu, es reiniciarà el procés de generació de zones i habitacions fins que totes les zones estiguin connectades correctament.

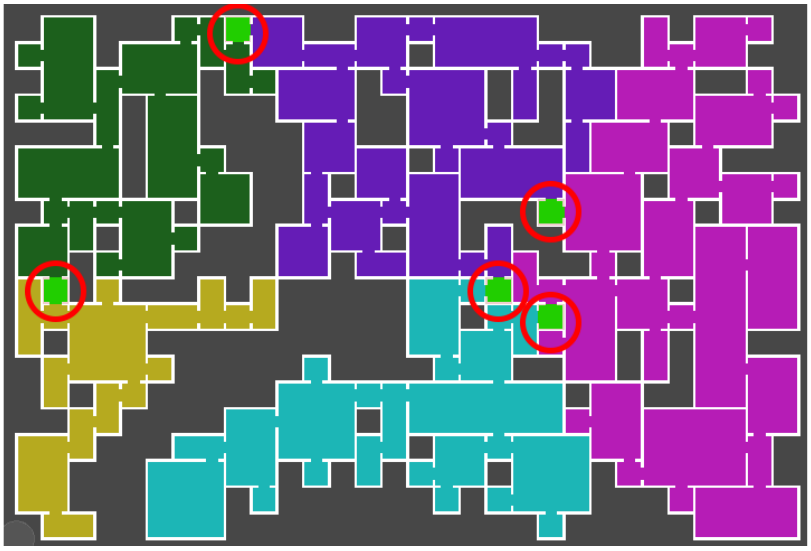

Figura 42. Exemple de generació de sales de transició. Font: Elaboració pròpia.

6.7.2. Sales de *boss*

El següent pas de l'script SpecialRoomsGenerator és implementar les sales de *boss*. Per fer-ho, s'agafen totes les sales que conformen una zona i se separen totes aquelles que compleixen una sèrie de requisits. Perquè una habitació sigui adient es mira la seva mida, la seva posició i els veïns que té. Finalment, se selecciona l'habitació més llunyana de la sala de transició de la zona prèvia i a ser possible amb el nombre de veïns més reduït. Les sales de *boss* estan representades amb la icona d'una calavera.

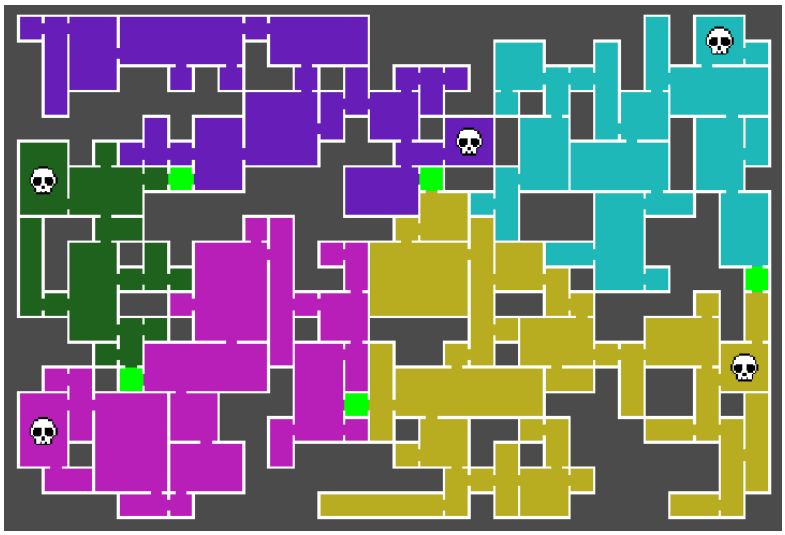

Figura 43. Exemple de sales de *boss*. Font: Elaboració pròpia.

6.7.3. Sales de guardar partida

La generació de les sales de guardar es fa en dos passos, el primer és col·locar la sala de guardar a prop de la sala de *boss* i el segon és col·locar la resta de sales de guardar de manera escampada per la zona que pertanyen. Aquest procés es farà per cada zona del mapa.

Tanmateix, com s'ha vist anteriorment, les sales de guardar han de complir quatre requisits:

- Han de ser de mida 1x1.
- Només es poden obrir per un dels laterals.
- Sempre hi ha d'haver una habitació de guardar a prop de la sala de *boss*.
- El nombre total de sales de guardar d'una zona és proporcional a la mida d'aquesta.

En aquest cas, el nombre de sales de guardar d'una zona es determina per una variable numèrica modificable des de l'inspector de Unity.

Un cop comença cada iteració de zona es busquen espais buits a on poder col·locar les sales de guardar i s'afegeixen a una llista de llocs possibles.

El primer pas consisteix a col·locar la primera habitació per guarda partida, aquesta serà la que està més a prop de la sala de *boss*. Per fer-ho s'ha buscat la localització de la sala de *boss* i s'ha ordenat la llista de llocs possibles per distància a aquesta, de tal manera que les habitacions que estan a sobre de la llista seran les que estan més a prop.

El segon pas consisteix a col·locar la resta de sales de guardar, per fer-ho es divideix la ruta d'aquella zona per tants cops com nombre de sales de guardar que queden per col·locar. Per últim, se selecciona una habitació aleatòria de cada secció de ruta.

Dividint la ruta en seccions aconseguim que les sales de guardar quedin més repartides per tota la zona i no totes concentrades a algun lloc específic.

Finalment, el resultat final són una sèrie d'habitacions de guardar partida representades de color vermell a les diferents zones del mapa.

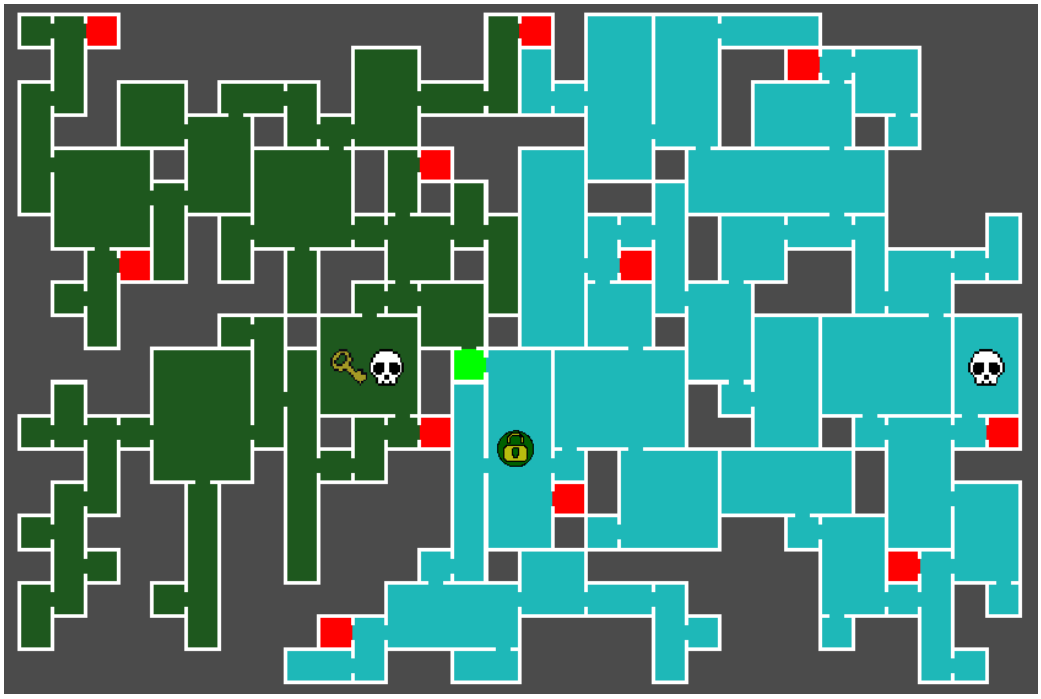

Figura 44. Exemple de sales de guardar. Font: Elaboració pròpia.

6.7.4. Sales de teleportació

De la mateixa manera que les sales de guardar partida, les sales de teleportació han de complir tres requisits:

- Han de ser de mida 1x1.
- Només es poden obrir pels laterals.
- Només n'hi pot haver una per Zona.

Per aquest projecte, en haver-hi un màxim de sis zones es va decidir col·locar dues sales de teleportació només en cas que el nombre de zones sigues cinc o superior, ja que d'altra banda, el nombre de zones era massa petit.

La primera sala de teleportació es col·locarà a una posició aleatòria de la primera o segona zona, la segona zona serà escollida buscant la zona més allunyada de la primera i que no sigui consecutiva a aquesta. Finalment, es buscarà un lloc a on col·locar la segona habitació de teleportació dins de la segona zona escollida.

Les sales de teleportació són representades amb color blau fosc.

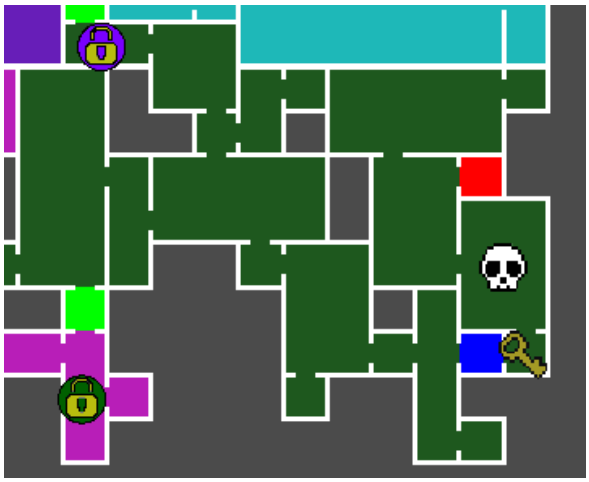

Figura 45. Exemple de zona amb sala de teleportació. Font: Elaboració pròpia.

6.7.5. Sales de porta

Les sales de porta o *gate rooms* són les sales que han d'evitar que el jugador pugui avançar lliurement pel mapa. Per tal de fer-ho es va decidir col·locar-les a prop de l'entrada de cada zona, d'aquesta manera es té la certesa que el jugador no pugui passar mai per un camí alternatiu.

Les sales de porta estan representades amb la icona d'un cadenat i el color de la zona que s'ha de superar abans per poder accedir a la nova zona.

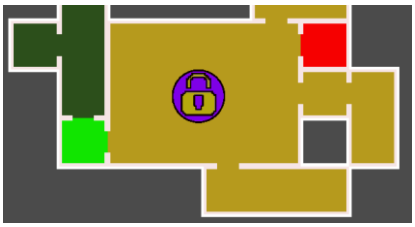

Figura 46. Exemple de sala porta. Font: Elaboració pròpia.

6.7.6. Sales clau

Aquestes són les habitacions a on el jugador obtindrà *l'item* o el poder necessari per desbloquejar les sales de porta que s'ha trobat anteriorment en el seu camí. Es va decidir col·locar-les dins de les sales de *boss* o a l'habitació de darrere en cas que fos una habitació sense sortida.

Les sales clau estan representades amb la icona d'una clau.

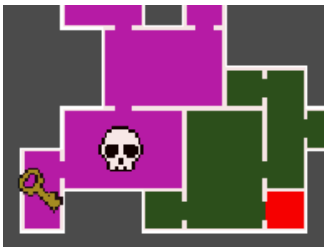

Figura 47. Exemple de sala clau. Font: Elaboració pròpia.

6.7.7. Diagrama de classe

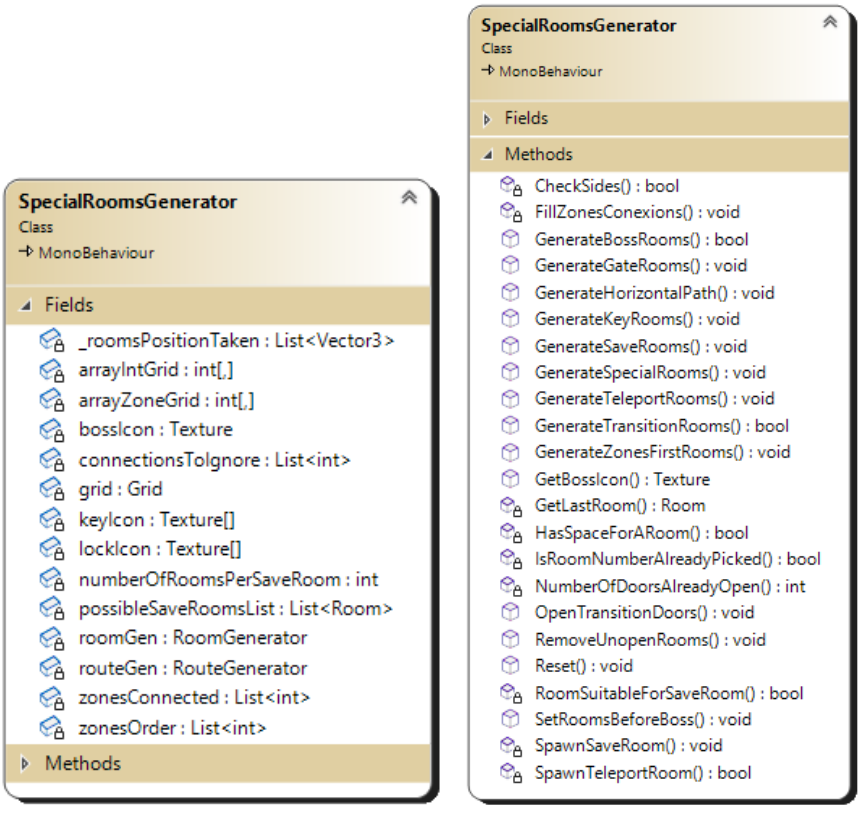

Figura 48. Diagrama de classe de SpecialRoomGenerator. Font: Elaboració pròpia.

6.8. Setena fase

En aquesta fase l'objectiu principal era desenvolupar un editor de Unity personalitzat i una eina per instanciar habitacions noves.

El motiu darrere d'aquesta fase és que l'editor de Unity s'adapti a les necessitats d'aquest projecte i que sigui més fàcil per un dissenyador modificar el mapa generat al seu gust.

6.8.1. Editor personalitzat de Unity

Per tal de facilitar l'edició de paràmetres o valors es van fer cinc scripts que modifiquen l'editor per defecte de Unity. Aquests cinc scripts s'encarreguen de modificar el comportament de l'editor, com per exemple, generar botons, espais o etiquetes.

S'han generat set botons nous per complir les següents funcions:

Boto	Descripció
Generate Grid	Genera la quadrícula en cas que no estigui creada.
Delete Grid	Esborra la quadrícula.
Generate Map	Genera el mapa d'acord amb els valors establerts.
Erase Map	Esborra el mapa.
Regenerate Zones	Torna a generar les zones.
Regenerate Rooms	Torna a generar les habitacions.

Generate Special Rooms	Genera les habitacions especials: sales de guardar, sales de transició, sales de teleportació, sales de boss, etc.

Taula 3. Descripció dels botons de l'editor personalitzat de Unity. Font: Elaboració pròpia.

A part dels botons generats pels scripts, l'usuari té l'opció de modificar alguns paràmetres del generador per adaptar-lo a les seves necessitats, alguns exemples són: escollir les dimensions de la quadrícula, el número de *walkers*, el nombre de zones, el color de cada zona, la ràtio de sales de guardar per habitacions o les icones a mostrar.

A més també hi ha *checks* per veure algunes opcions de *debug* o *testing*, com per exemple poder fer aparèixer i desaparèixer la quadrícula, veure l'assignació de número per habitació, veure el gràfic o la ruta generada.

6.8.2. Eina personalitzada d'habitacions

Per encara facilitar més el disseny s'ha creat una eina personalitzada amb l'objectiu principal de generar habitacions. Aquesta eina permet triar diversos paràmetres per tal de generar el tipus d'habitació desitjada. Els paràmetres són els següents:

Paràmetre	Descripció
Zone number	Permet escollir la zona de la nova sala que es vol generar.
Manual/Automatic room number	*Check* que permet escollir si assignar un número específic a la nova habitació o generar-ne un automàticament.
Room Width	Número referent a l'amplada que tindrà la nova habitació.

Room Height	Número referent a l'alçada que tindrà la nova habitació.
Is a spacial room	*Check* per escollir si la nova sala forma part del conjunt de sales especials. En cas que sí, s'obrirà una llista de *checks* per escollir més en detall el tipus i els paràmetres desitjats.

Taula 4. Paràmetres de l'eina Create Room. Font: Elaboració pròpia.

Quan tots els paràmetres de la nova sala estan especificats, l'usuari podrà clicar al botó de *Create Room*. En aquest moment les RoomCells que componen l'habitació es generaran en el mapa i l'usuari serà lliure de col·locar l'habitació a on desitgi, finalment haurà de fer clic en el botó de *Regenerate Rooms* per col·locar-la definitivament al lloc escollit i generar la nova ruta d'habitacions.

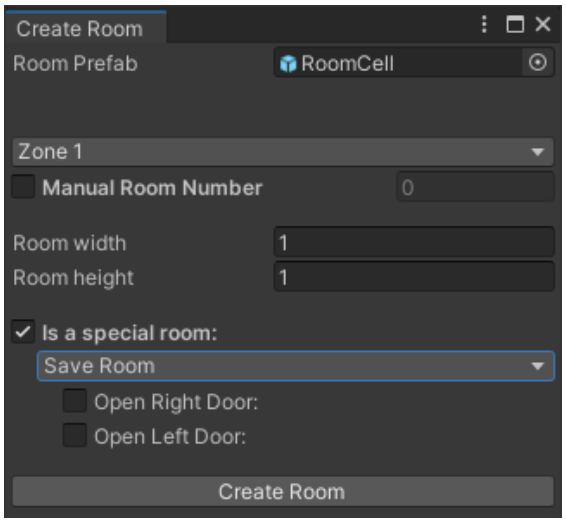

Figura 49. Eina Create Room. Font: Elaboració pròpia.

Per últim, s'ha personalitzat l'editor de Unity per tal que quan se selecciona una habitació pugui ser esborrada mitjançant un botó anomenat *Delete Room*.

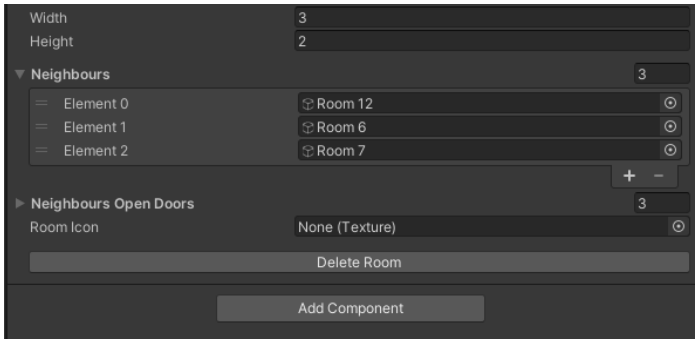

Figura 50. Botó *Delete Room*. Font: Elaboració pròpia.

6.8.3. Diagrames de classe

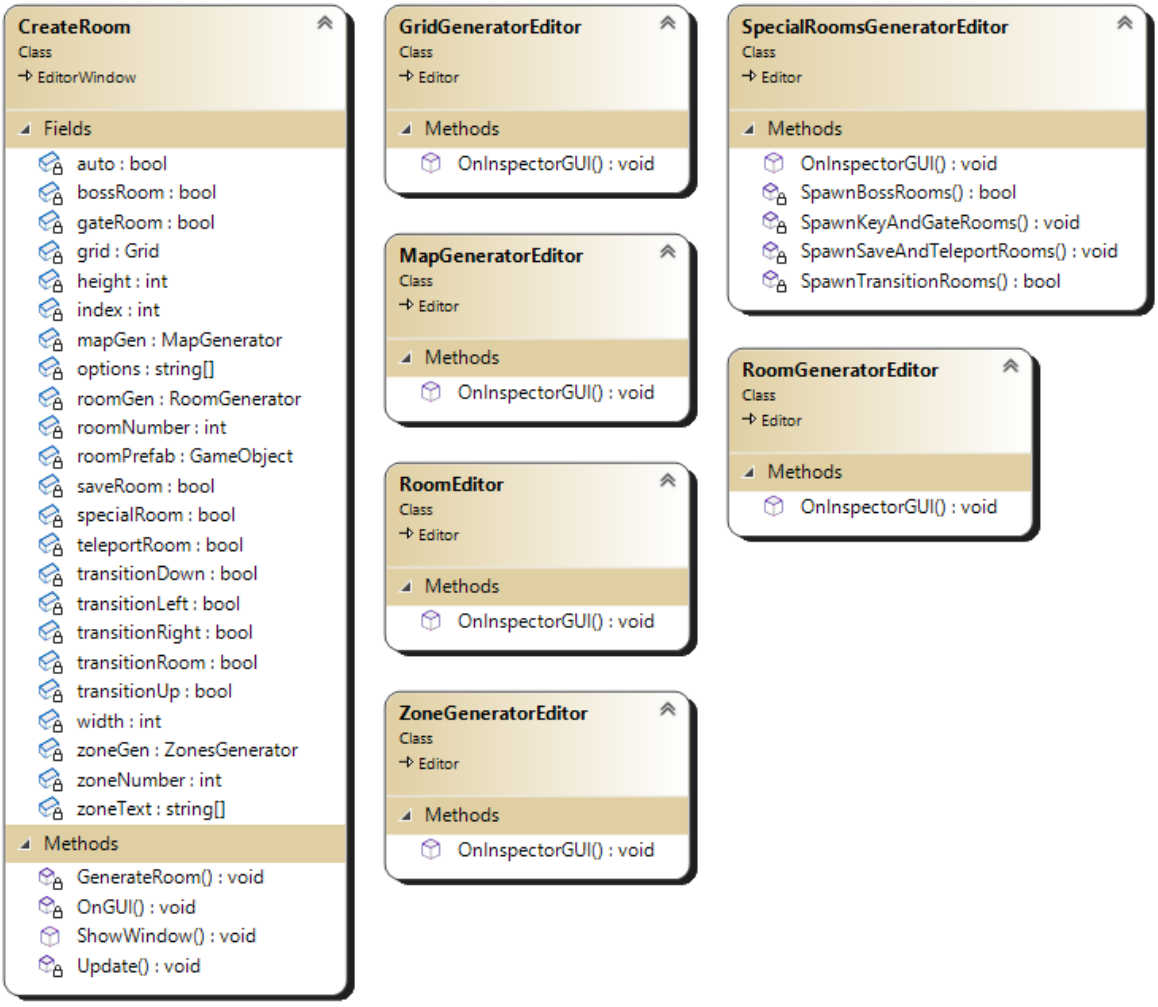

Figura 51. Diagrama de classes CreateRoom, GridGeneratorEditor, MapGeneratorEditor, ZoneGeneratorEditor, RoomGeneratorEditor, SpecialRoomsGeneratorEditor i RoomEditor. Font: Elaboració pròpia.

7. Anàlisis dels resultats

En aquesta secció es mostren els resultats finals obtinguts al llarg del treball i s'analitzen el funcionament i el rendiment de l'eina desenvolupada.

7.1. Rendiment

Cal destacar que depenent de la màquina amb la qual es treballa els resultats poden variar. Per aquest motiu és important mencionar les especificacions de l'ordinador amb el qual s'ha treballat durant les proves de rendiment:

Operating System	Windows 10 Pro 64-bit (10.0, Build 19043) (19041.vb_release.191206-1406)
Processor	Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz (8 CPUs), ~2.8GHz
Memory	16384MB RAM
Graphics Card	NVIDIA GeForce GTX 1070

7.1.1. Taula de resultats

La taula següent mostra la mitjana aritmètica dels resultats obtinguts durant la fase de proves de rendiment, cada test s'ha realitzat un total de deu vegades.

Mida Mapa (cel·les)	Número de Walkers	Mitjana Temps (segons)	Mitjana FPS (aprox)
100	80	1,87	80
300	240	4,61	53
600	400	10,38	41
1000	800	19,7	25

Taula 5. Taula de resultats. Font: Elaboració pròpia.

7.1.2. Gràfic de rendiment

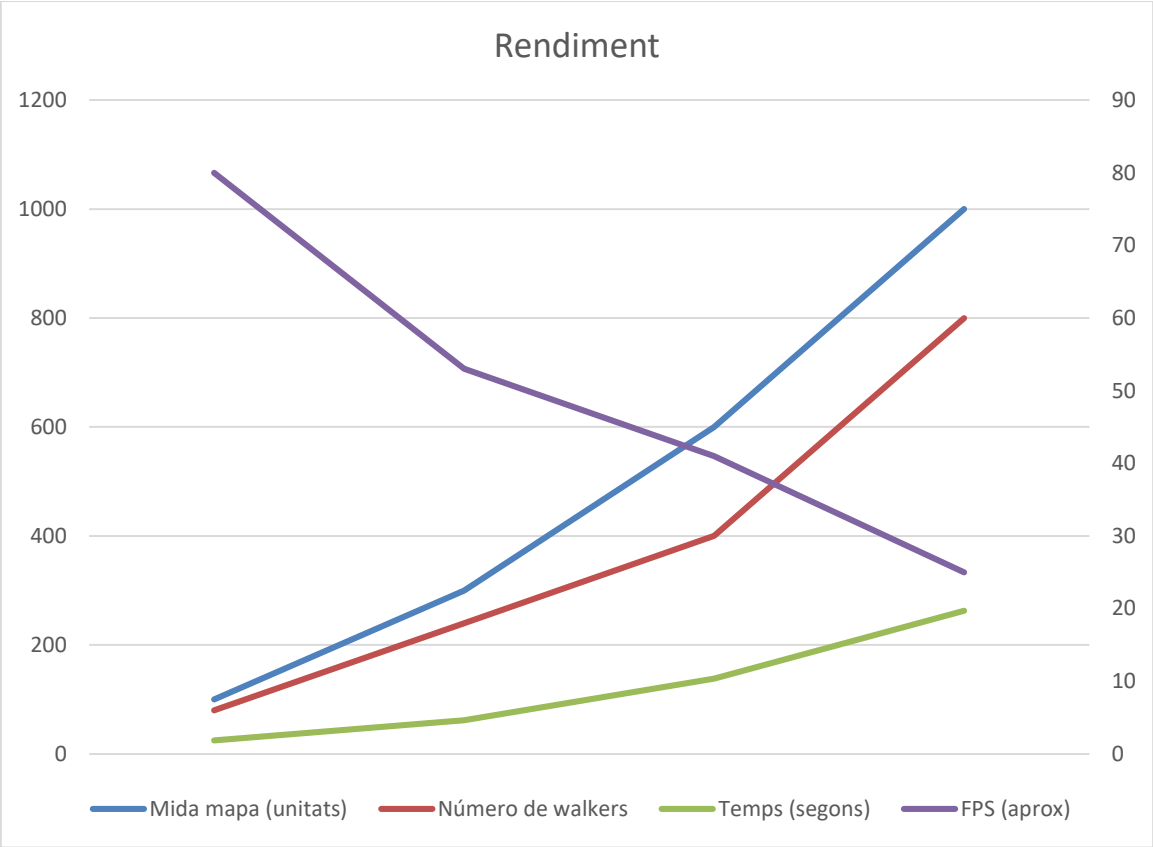

Figura 52. Gràfic de rendiment. Font: Elaboració pròpia.

Els valors de l'esquerra (0-1200), estan relacionats amb la mida del mapa i el número de walkers. Els valors de la dreta (0-90), estan relacionats amb el temps de generació i els FPS (*frames* per segon).

7.1.3. Anàlisis del gràfic

Com es pot apreciar en el gràfic, el rendiment baixa considerablement quan s'intenten generar mapes grans amb un número elevat de walkers, mentre que el temps de generació augmenta proporcionalment a la mida i el nombre de walkers.

Cal destacar que la mida del mapa no afecta gairebé gens en el rendiment, en contra part, la quantitat de walkers sí que té un gran impacte, per aquest motiu, si es genera un mapa de grans dimensions amb un nombre reduït de walkers, el

resultat serà bastant més alt en rendiment però el temps d'execució es veurà afectat considerablement.

Es pot veure un exemple de com evoluciona el temps d'execució i el rendiment general si es compara la primera fila de la taula de resultat i l'última. Amb una mida de mapa de cent cel·les i amb vuitanta walkers s'aconsegueix generar un mapa en una mitjana de 1,87 segons i una mitjana de rendiment de vuitanta FPS, mentre que amb una mida de mapa de mil cel·les i vuit-cents walkers la mitjana de temps és de 19,70 segons i la mitjana de *frames* per segon és de vint-i-cinc. Això vol dir que en un mapa deu vegades més gran que l'original, el temps de generació incrementa en 10,5 mentre que el rendiment en FPS es divideix en 3,2.

Un punt a tenir en compte és que un cop ha acabat la generació del mapa, el rendiment millora considerablement i els valors són més acceptables quan es fa *playtesting* del mapa.

7.2. Anàlisis de mapa

Els mapes obtinguts pel generador són tancats, interconnectats, variats i modificables. A més es té la llibertat d'escollir el número de zones que formen el mapa i cada zona té un conjunt de sales especials que són clau pel gènere de Metroidvania.

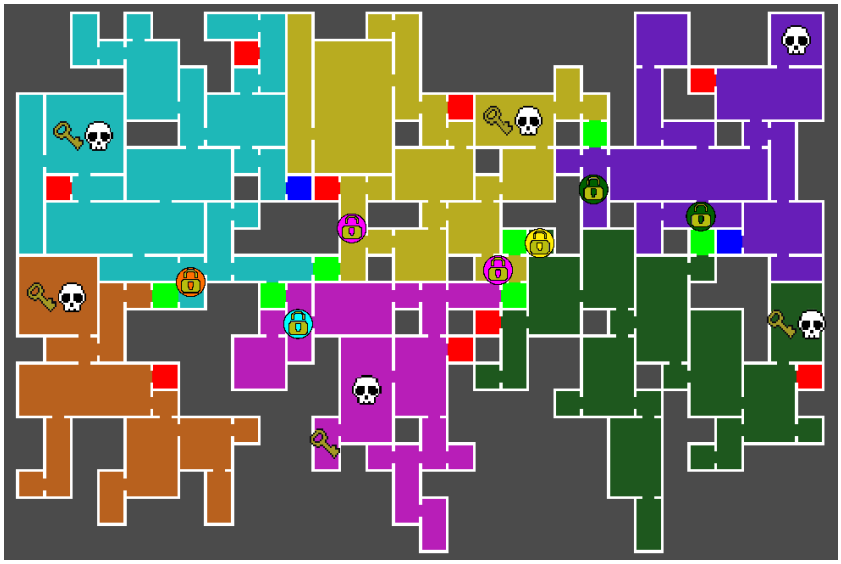

Figura 53. Exemple de mapa generat procedimentalment. Font: Elaboració pròpia.

8. Conclusions i reflexió

Finalment, en aquest últim apartat del projecte s'expressaran les conclusions i reflexions finals del projecte desenvolupat.

8.1. Conclusions generals

En general, el projecte ha sigut satisfactori. S'ha complert amb el cronograma establert, s'han aconseguit complir els objectius principals proposats i ha permès obtenir un aprenentatge que de no haver dut a terme el treball, no s'hauria assolit.

Tot i això, com es mostra a l'apartat d'anàlisis dels resultats, el rendiment i el temps de generació són millorables. A més tampoc s'han pogut implementar tots els objectius secundaris a causa de la falta de temps.

En un futur, és possible acabar d'implementar tots els objectius secundaris i mirar de millorar l'eina per acabar de tenir un projecte tancat i més polit.

8.2. Objectius principals

Com s'ha pogut veure, s'ha aconseguit desenvolupar un generador de mapes procedimental d'estil Metroidvania funcional. Tot i que el generador compleix amb tots els objectius principals proposats, no és del tot òptim i es podria millorar molt, sobretot a l'apartat del temps de generació i rendiment general de l'eina.

Durant el desenvolupament es va decidir donar prioritat a la funcionalitat i visualització per sobre de l'optimització i bones pràctiques en diverses ocasions, ja que la finalitat d'aquest treball no era fer una eina llesta per ser llençada al mercat o per ser implementada en un joc real, sinó descobrir si era possible desenvolupar un generador de mapes procedimentals 2D que pogués generar mapes d'estil Metroidvania tal com ho fan els jocs del gènere Roguelike.

Per aquest motiu, com ja s'ha mencionat anteriorment, es podrien millorar molts apartats del generador, com per exemple substituir el mètode de DLA (Diffusion Limited Aggregation) pel mètode RRP (Random Room Placement) o directament suprimir el moviment aleatori dels walkers i fer que es generin directament a un lloc específic tal com s'ha mencionat a l'apartat de

desenvolupament i resultats [6.3]. D'aquesta manera s'aconseguiria una gran reducció en el temps de generació i un augment de rendiment molt significatiu, ja que, el que afecta més al rendiment són totes les crides de la funció Update de cada walker per tal de poder-se moure dins de la quadrícula i comprovar si han topat contra el nucli central o no.

Un altre punt a millorar i que també s'ha mencionat, són les llistes de zona, que en comptes de crear-se manualment amb un número definit, es podrien generar dinàmicament per així adaptar-se a cada situació possible i evitar gastar recursos fútilment.

Dit això, l'habitació modular que s'ha fet servir per la construcció de les habitacions ha resultat ser bastant versàtil i fàcil de fer servir. A més, fa que el projecte sigui molt lleuger, ja que amb un únic prefab s'han pogut generar totes les habitacions del mapa.

Respecte al segon punt proposat com a objectiu principal, també s'ha aconseguit realitzar de manera satisfactòria. El generador és una eina amigable, fàcil d'usar, ofereix opcions per modificar el comportament del generador i es poden afegir o eliminar habitacions del mapa en qualsevol moment.

Tanmateix, també es podria millorar més, per exemple, el fet que s'han d'especificar quines portes s'han d'obrir en el cas de crear algunes de les sales especials o el fet que totes les habitacions s'hagin de reestructurar quan hi ha una modificació en el mapa, per petita que sigui.

8.3. Objectius secundaris

Pel que fa als objectius secundaris, no s'han pogut complir tots a causa de la falta de temps. No obstant això, s'ha pogut implementar una versió del jugador molt bàsica amb el que l'usuari o dissenyador es podrà moure i navegar pel mapa generat per tal de realitzar un playtesting bàsic.

Pel que fa al tema de poder importar i exportar mapes ja generats, es podria mirar d'implementar en un futur mitjançant un sistema de seed, ja que d'aquesta forma seria molt fàcil de gestionar la transferència de mapes entre usuaris.

Per últim, es podrien afegir més opcions de personalització per tal de tenir una eina més completa, com per exemple poder especificar el nombre d'enemics finals per zona o el nombre d'habitacions màxims per zona. També es podria mirar d'implementar que les opcions de personalització es poguessin importar al generador amb fitxers json o xml.

9. Bibliografia/Referències

9.1. Referències

A* Pathfinding Project Pro | AI | Unity Asset Store. (s.d.). Recuperat 10 juny 2022, de https://assetstore.unity.com/packages/tools/ai/a-pathfinding-project-pro-87744?aid=1011l7JId&utm_campaign=unity_affiliate&utm_medium=affiliate&utm_source=partnerize-linkmaker

Baron, J. (2017). Procedural Dungeon Generation Analysis and Adaptation.

Bissell, T. (2010). Extra Lives: Why Video Games Matter. Knopf Doubleday Publishing Group.

Boehm, B. W. (1988). A spiral model of software development and enhancement. Computer, 21(5), 61-72. https://doi.org/10.1109/2.59

Bourke, P. (2006). Constrained diffusion-limited aggregation in 3 dimensions. Computers & Graphics, 30(4), 646-649. https://doi.org/10.1016/j.cag.2006.03.011

Bucklew, B. (2017). Algorithms and Approaches. En T. X. Short & T. Adams, Procedural Generation in Game Design. Taylor & Francis Group.

Bycer, J. (2018). 20 Essential Games to Study. CRC Press.

Cossu, S. M. (2019). Game Development with GameMaker Studio 2: Make Your Own Games with GameMaker Language. Apress.

GameSquid. (2020, abril 19). Binding of Isaac: Room Generation Explained! https://www.youtube.com/watch?v=1-HIA6-LBJc

Graham, R. L., & Hell, P. (1985). On the History of the Minimum Spanning Tree Problem. IEEE Annals of the History of Computing, 7(1), 43-57. https://doi.org/10.1109/MAHC.1985.10011

Grey, D. (2017). When and Why to Use Procedural Generation. En T. X. Short & T. Adams, Procedural Generation in Game Design. Taylor & Francis Group.

Gutiérrez Rodríguez, A., Cotta, C., & J. Fernández Leiva, A. (2018). An Evolutionary Approach to Metroidvania Videogame Design.

Hanson, C. (2018). Game Time: Understanding Temporality in Video Games. Indiana University Press.

Harris, J. (2021). Exploring Roguelike Games. CRC Press.

Kalata, K. (2017). Hardcore Gaming 101: Castlevania. Game Press.

Kok, B. (2021). Beginning Unity Editor Scripting: Create and Publish Your Game Tools. Apress.

Kumar, B., Kumar, U., Kumar, S., & Tomer, V. (2019). Comparison and Performance Evaluation of Boundary Fill and Flood Fill Algorithm. International Journal of Innovative Technology and Exploring Engineering, 8(12s3), 9-13. https://doi.org/10.35940/ijitee.L1002.10812S319

Minkkinen, T. (2016). Basics of Platform Games.

Moana Thompson, K. (2006). Scale, Spectacle and Movement: Massive Software and Digital Special Effects in The Lord of The Rings. En From Hobbits to Hollywood: Essays on Peter Jackson's Lord of the Rings (p. 403). Rodopi.

Na, S., Xumin, L., & Yong, G. (2010). Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm. 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, 63-67. https://doi.org/10.1109/IITSI.2010.74

Petruzzi, J. (2018, juliol 9). Chasm Hits PS4, PS Vita July 31. PlayStation.Blog. https://blog.playstation.com/2018/07/09/chasm-hits-ps4-ps-vita-july-31/

Prim, R. C. (1957). Shortest Connection Networks And Some Generalizations. Bell System Technical Journal, 36(6), 1389-1401. https://doi.org/10.1002/j.1538-7305.1957.tb01515.x

Shaker, N., Togelius, J., & Nelson, M. J. (2016). Procedural content generation in games. Springer Berlin Heidelberg.

Stalnaker, T. (2020). Procedural Generation of Metroidvania Style Levels. 73.

van der Linden, R., Lopes, R., & Bidarra, R. (2014). Procedural Generation of Dungeons. IEEE Transactions on Computational Intelligence and AI in Games, 6(1), 78-89. https://doi.org/10.1109/TCIAIG.2013.2290371

Watkins, R. (2016). Procedural Content Generation for Unity Game Development. Packt Publishing Ltd.

Witten, T. A., & Sander, L. M. (1981). Diffusion-Limited Aggregation, a Kinetic Critical Phenomenon. Physical Review Letters, 47(19), 1400-1403. https://doi.org/10.1103/PhysRevLett.47.1400

9.2. Obres audiovisuals

ArtPlay. (2019). Bloodstained: Ritual of the Night [Windows, Nintendo Switch, Playstation 4, Xbox One].

Bethesda Game Studios. (2011). The Elder Scrolls V: Skyrim [Microsoft Windows, PlayStation 3, Xbox 360, PlayStation 4, Xbox One, Nintendo Switch, PlayStation 5, Xbox Series X/S].

Bit Kid, Inc. (2018). Chasm [Windows, macOs, Linux, Nintendo Switch, Playstation 4, Xbox One, Playstation Vita].

Blizzard Entertainment. (1997). Diablo [Microsoft Windows, Classic Mac OS, macOS, PlayStation, PlayStation 3, PlayStation 4, Xbox 360, Xbox One, Nintendo Switch, PlayStation 5, Xbox Series X/S].

Capcom. (1985). Ghosts ’n Goblins [Commodore 64, 16, ZX Spectrum, Amstrad CPC, NES (1986)]. Capcom.

Gearbox Software. (2009). Borderlands [Windows, macOS, Playstation 3, Xbox 360].

Hiraoka, H., & Watanabe, D. (2015). Dreeps: Alarm Playing Game [Nintendo 3DS, iOS devices].

Interactive Data Visualization, Inc. (2002). SpeedTree [Windows, Xbox One, Xbox 360, PlayStation 4, PlayStation 3, PlayStation Vita, Mac OS X, Linux].

Key, E., & Kanaga, D. (2013). Proteus [Microsoft Windows, OS X, Linux, PlayStation 3, PlayStation Vita].

Konami. (1997). Castlevania: Symphony of the Night [Playstation, Saturn]. Konami.

McMillen, E. & Himsl, Florian. (2011). The Binding of Isaac [Windows, macOs, Linux].

Moon Studios. (2015). Ori Saga [Windows, Xbox One, Nintendo Switch].

Morningstar Game Studio. (2017). A Robot Named Fight! [Windows, macOs, Linux, Nintendo Switch, Playstation 4, Xbox One, Playstation Vita].

Motion Twin. (2018). Dead Cells [Windows, Nintendo Switch, Playstation 4, Xbox One, macOs, Linux, iOS, Android].

Nintendo Research & Development 1. (1994). Super Metroid [SNES]. Nintendo.

Supergiant Games. (2020). Hades [Windows, Nintendo Switch, Playstation 4, Xbox One, macOs, Playstation 5, Xbox Series X/S].

Team Cherry. (2017). Hollow Knight [Windows, macOs, Linux, Nintendo Switch, Playstation 4, Xbox One].

The Game Kitchen. (2019). Blasphemous [Windows, Nintendo Switch, Playstation 4, Xbox One].

.theprodukkt. (2004). .Kkrieger [Microsoft Windows].

Thunder Lotus Games. (2017). Sundered [Windows, macOs, Linux, Nintendo Switch, Playstation 4, Xbox One, Stadia].

Toy, M., & Wichman, G. (1980). Rogue.

TecnoCampus

Centres universitaris adscrits a la

upf. Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

Generació procedimental de mapes 2D d'estil Metroidvania

ANNEX

Albert Abulí Federico
Tutor: Rafael González Fernández
2021-2022

TecnoCampus Mataró-Maresme

Índex

//

Índex de figures

1. Localització de continguts

Tot seguit es detallarà la localització de tot el contingut digital entregat en aquest projecte. Totes les rutes són a partir de la carpeta base del google drive anomenada:

Abulí_Federico_GeneracióProcedimentalMapes2DMetroidvania_TFG

1.1. Codi font

Projecte de Unity comprimit en zip.

- **Ruta**: /Annex/Projecte/

1.2. Vídeo demostratiu

Vídeo demostratiu de l'eina desenvolupada a Unity.

- **Ruta**: /Annex/Vídeo demostratiu/Abulí_Federico_TFG_Vídeo.mp4

1.3. Documentació

Documentació principal del treball. Està situada a la carpeta base del google drive:

- **Ruta:** Abulí_Federico_GeneracióProcedimentalMapes2DMetroidvania_TFG/

1.4. Annex

Carpeta a on es troba tot el contingut digital a excepció de la documentació principal.

- **Ruta**: /Annex/

2. Manual d'instruccions

A continuació es detalla com operar el generador procedimental de mapes 2D d'estil Metroidvania desenvolupat al llarg del projecte.

2.1. Visió general de l'eina

En obrir el projecte de Unity es pot veure una escena buida acompanyada de tres components a la pestanya de *Hierarchy* de Unity. Aquests tres components són la càmera principal, un GameManager i el MapGenerator:

2.2. Generador de mapes

El MapGen o generador de mapes està format per sis scripts diferents, cada un encarregat d'una tasca concreta.

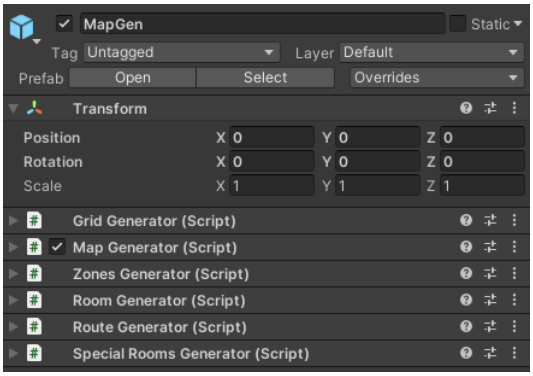

Figura 1. Scripts del generador de mapes. Font: Elaboració pròpia.

2.2.1. Grid Generator

El GridGenerator serveix per delimitar la mida de la quadrícula que es fa sevir per generar el mapa. Els valors es poden modificar per ajustar la mida del mapa.

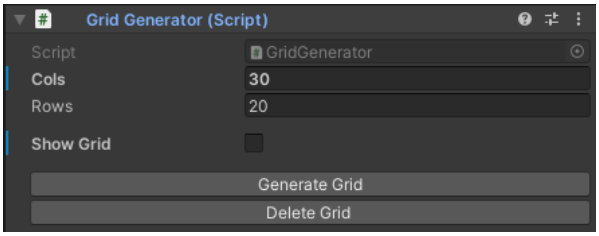

Figura 2. Vista de l'script Grid Generator. Font: Elaboració pròpia.

- **Cols**: Serveix per indicar el número de columnes de la quadrícula.
- **Rows**: Serveix per indicar el número de files de la quadrícula.
- ***Check* Show Grid**: Aquesta casella serveix per ensenyar o amagar la graella del mapa:

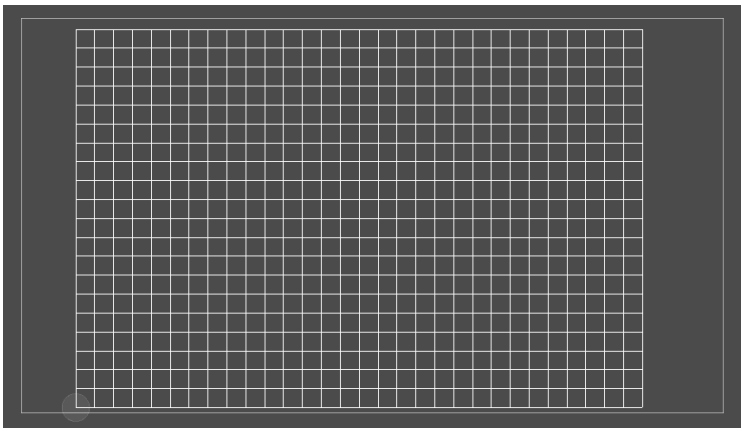

Figura 3. Quadrícula del GridGenerator. Font: Elaboraicó pròpia.

Al final del GridGenerator hi ha dos botons:

- **Botó Generate Grid**: Serveix per construir la graella en cas que no ho estigui.
- **Botó Delete Grid**: Serveix per eliminar la quadrícula.

2.2.2. Map Generator

El següent script s'anomena MapGenerator i és l'encarregat de començar la generació del mapa, aquesta tasca inclou: *l'spawn* dels walkers per delimitar la forma del mapa, la generació de zones, la generació d'habitacions i la generació de rutes.

En aquest script es pot modificar el següent element:

- **Number of Walkers**: Indica el número de cèl·lules o walkers que formaran el mapa.

A sota es poden veure dos botons:

- **Botó Generate Map**: Serveix per començar la generació.
- **Botó Erase Map**: Serveix per eliminar el mapa generat.

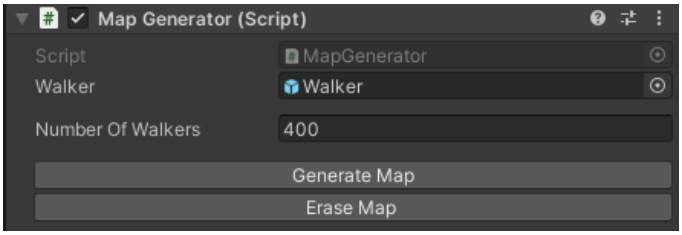

Figura 4. Vista de l'script MapGenerator. Font: Elaboració pròpia.

2.2.3. Zone Generator

L'script ZoneGenerator s'encarrega de generar les zones. En aquest script es poden indicar diverses opcions per personalitzar el resultat de les zones que s'obtindran en generar el mapa. Hi ha quatre elements de personalització i un botó:

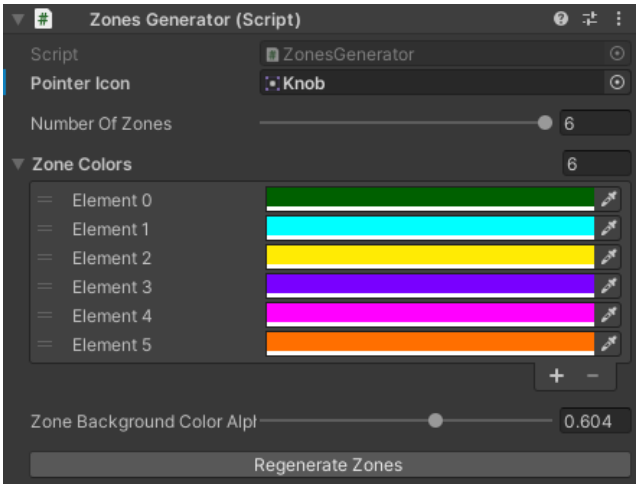

Figura 5. Vista de l'script ZoneGenerator. Font: Elaboració pròpia.

- **Pointer Icon**: serveix per indicar la icona dels *centroids* o *pointers* de cada grup generat per l'algorisme K-means.
- **Number of zones**: Serveix per delimitar el nombre de zones que es crearan. Per aquest projecte es pot seleccionar entre una sola zona i sis com a màxim.

- **Zone colors**: Serveix per representar les zones amb un color determinat. Ajuda a distingir-les visualment.

- **Zone Background color alpha**: Serveix per controlar l'opacitat del color de les zones.

- **Botó Regenerate Zones**: la seva funció és regenerar les zones en cas que el resultat obtingut no sigui satisfactori.

2.2.4. Room Generator

Aquest script s'encarrega de generar les habitacions i ordenar el resultat obtingut a la pestanya de *Hierarchy*. Hi ha quatre elements:

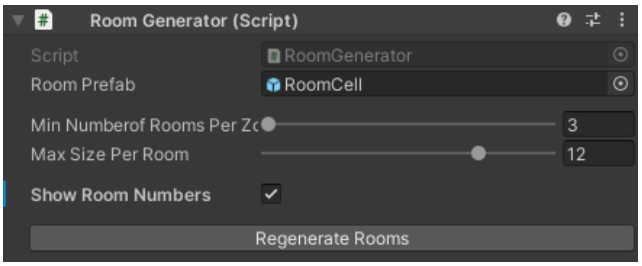

Figura 6. Vista de l'script RoomGenerator. Font: Elaboració pròpia.

- **Min number of rooms per zone**: Serveix per indicar el nombre mínim d'habitacions que ha de tenir una zona. Si aquest paràmetre no es compleix, les zones es regeneraran fins que ho faci.

- **Max size per room**: limita la mida de les habitacions quan el generador està fusionant les columnes. Si la mida és molt petita, el resultat final seran les columnes sense fusionar. Si el valor és modificat un cop el mapa ha estat generat, s'hauran de regenerar les zones per veure el canvi.

- **Show room numbers**: Mostra el mapa de números assignats a cada habitació:

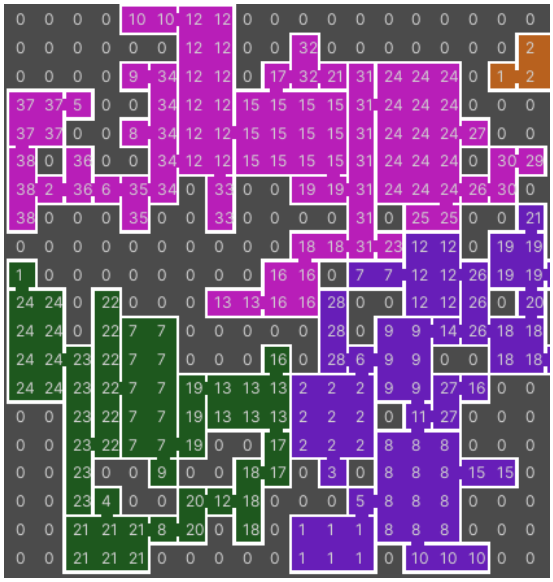

Figura 7. Mapa de números d'habitacions. Font: Elaboració pròpia.

- **Botó Regenerate Rooms**: Serveix per regenerar les habitacions. S'utilitza per reestructurar les habitacions quan hi ha hagut un canvi en el mapa, siguin habitacions noves o sales eliminades.

2.2.5. Route Generator

La funció d'aquest script és generar les rutes de cada zona i l'ordre en què el jugador haurà de travessar-les. Té tres elements a destacar:

- **Zone Order**: Mostra l'ordre final de les zones en forma de llista numèrica.
- **Show Graph**: Mostra el gràfic de nodes de cada zona.
- **Show Min route**: Mostra la ruta feta amb l'algorisme de Prim.

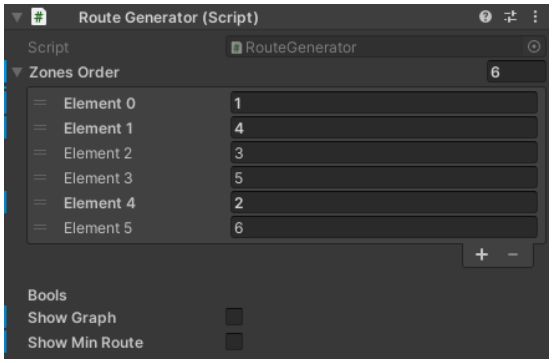

Figura 8. Vista de l'script RouteGenerator. Font: Elaboració pròpia.

2.2.6. Special Rooms Generator

Finalment, aquest script serveix per a la generació de les sales especials. En aquest projecte el tipus de sales especials generades són: sales de transició de zona, sales de guardar partida, sales de teleportació, sales de *boss* o enemic final de zona, sales de porta o *gate rooms* i sales de clau.

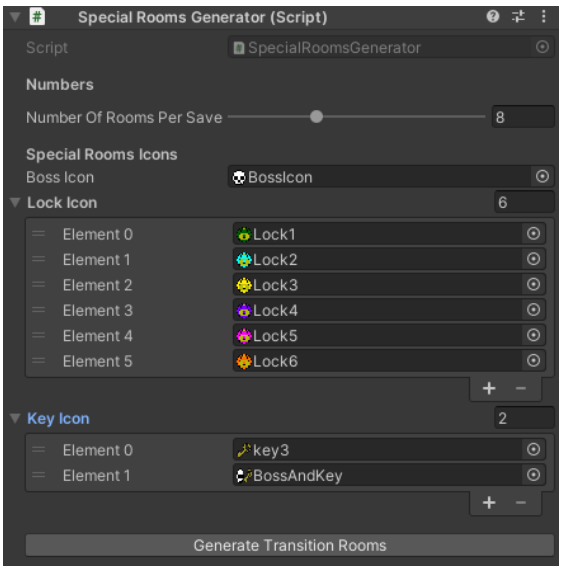

Figura 9. Vista de l'script SpecialRoomGenerator. Font: Elaboració pròpia.

Aquest script té cinc elements a destacar:

- **Number of rooms per save room**: Serveix per especificar la ràtio de número de sales de guardar per zona.
- **Boss icon**: Serveix per indicar la icona de les sales de *boss*.

- **Lock icon**: Serveix per indicar la icona de les sales de porta.
- **Key icon**: Serveix per indicar la icona de les sales clau.
- **Botó Generate Transition Rooms**: Serveix per generar les habitacions especials.

2.3. Eliminar habitacions

Per eliminar alguna habitació se seleccionarà mitjançant la pestanya de *Scene* o la pestanya de *Hierarchy*. Un cop seleccionada l'habitació, es ressaltarà d'un color fosc per mostra visualment quina habitació està seleccionada:

Figura 10. Exemple d'habitació seleccionada. Font: Elaboració pròpia.

Si després es mira la pestanya de l'inspector, es podrà veure la informació d'aquella sala:

Figura 11. Exemple de fitxa de sala. Font: Elaboració pròpia.

En aquesta fitxa hi trobem set elements a destacar:

- **Room number**: Indica el número assignat a l'habitació.
- **Width**: Indica l'amplada de l'habitació en caselles de la quadrícula.
- **Height**: Indica l'alçada de l'habitació en caselles de la quadrícula.
- **Neighbours**: Llista a on es veuen tots els veïns d'aquella sala.
- **Neighbours open doors**: Llista dels veïns que tenen una porta oberta a aquesta habitació.
- **Room icon**: Icona de l'habitació, algunes sales especials en fan ús.
- **Botó Delete Room**: Serveix per eliminar l'habitació. Cal puntualitzar que un cop eliminada la sala, s'hauran de regenerar les habitacions per tancar la porta oberta de les sales veïnes.

2.4. Afegir habitacions

Per tal d'afegir habitacions s'haurà d'accedir a una finestra nova desenvolupada per fer aquesta tasca, per accedir-hi es clicarà a la pestanya anomenada "Tools" en el menú superior de Unity i tot seguit, a l'opció "Create Room":

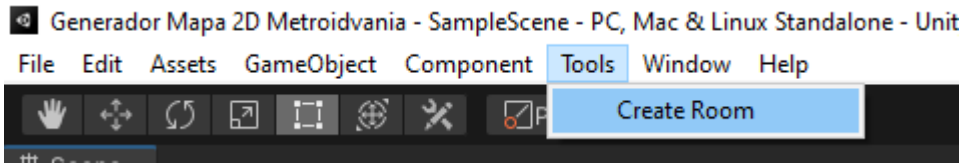

Figura 12. Obrir la finestra per afegir habitacions noves. Font: Elaboració pròpia.

En clicar, s'obrirà una finestra nova a on es podrà especificar el tipus d'habitació que es vol generar:

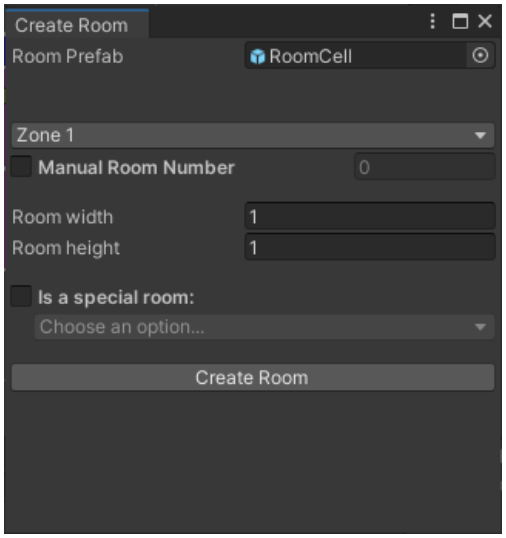

Figura 13. Eina de creació d'habitacions. Font: Elaboració pròpia.

Els elements configurables de la nova habitació són els següents:

- **Selecció de zona**: La primera opció és una llista desplegable de totes les zones, serveix per escollir la zona que formarà part la nova habitació.
- **Manual room number**: Casella de check que serveix per triar si assignar manualment la id de la sala en cas d'estar desseleccionada o introduir el número manualment.
- **Room width**: L'amplada de la nova habitació. El número representa les cel·les que ocuparà de la quadrícula horitzontalment.
- **Room height**: Alçada de la nova habitació. El número també indica les cel·les que ocuparà de la quadrícula verticalment.
- **Is a special room**: Casella check que serveix per indicar si la nova sala serà especial o no. En cas de ser-ho apareixeran algunes opcions més de configuració, com per exemple el tipus de sala especial o la direcció per on es vol obrir la nova sala.
- **Botó Create Room**: Serveix per crear la nova sala segons els paràmetres especificats.

Un cop clicat el botó de generar la nova habitació ens apareixerà a la pestanya de Scene. Allà es podrà moure per col·locar-la al lloc desitjat. Finalment, s'haurà de clicar al botó de Regenerate Rooms per acabar d'integrar-la en el mapa.

2.5. Playtesting

Un cop generat el mapa, es pot clicar al botó de *play* de Unity per tal de provar el mapa. Es crearà una pilota vermella capaç de navegar entre les habitacions del mapa.

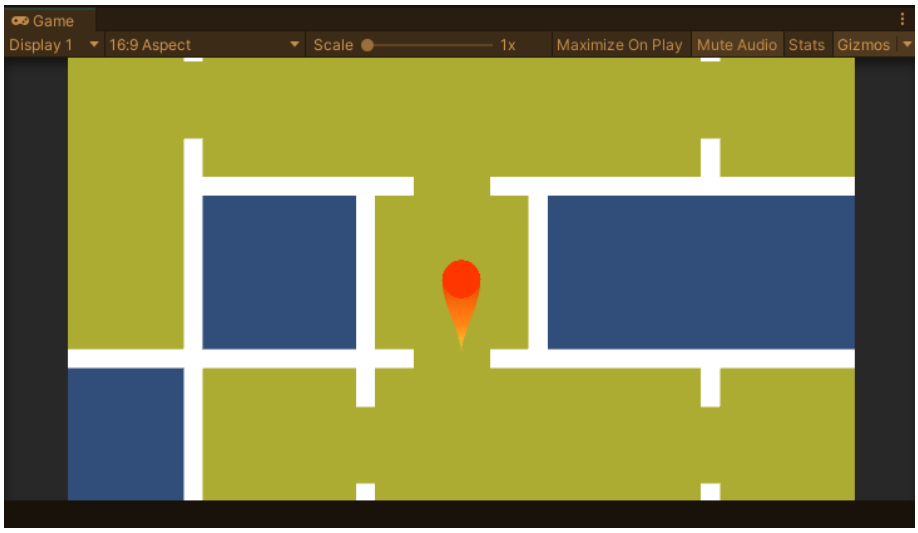

Figura 14. Vista del playtesting. Font: Elaboració pròpia.