TecnoCampus

Centres universitaris adscrits a la

Universitat Pompeu Fabra Barcelona

Grau en Disseny i Producció de Videojocs

DESENVOLUPAMENT D'UN FRAMEWORK PER GENERAR EDIFICACIONS PROCEDIMENTALMENT BASAT EN GRAMÀTIQUES

MEMÒRIA

Marçal Tineo Estevez

Tutor: Dr. Enric Sesa i Nogueras

2022-2023

TecnoCampus Mataró-Maresme

Abstract

The objective of this work is to develop a framework for the Unity engine that allows the user to create and modify buildings from a generative model based on grammars. Along with an interface for Unity to easily interact with grammars and set the materials and final appearance of the building.

Resum

L'objectiu d'aquest treball és desenvolupar un framework per al motor Unity que permeti a l'usuari crear i modificar edificacions a partir d'un model generatiu basat en gramàtiques. Juntament amb una interfície per a Unity per interactuar fàcilment amb les gramàtiques i establir els materials i aparença final de l'edifici.

Resumen

El objetivo de este trabajo es desarrollar un framework para el motor Unity que permita al usuario crear y modificar edificaciones a partir de un modelo generativo basado en gramáticas. Junto con una interfaz para Unity para interactuar fácilmente con las gramáticas y establecer los materiales y apariencia final del edificio.

Índex

0. GLOSSARI DE TERMES 1

1. INTRODUCCIÓ 4

2. MARC TEÒRIC 5

2.1. GENERACIÓ PROCEDIMENTAL DE CONTINGUTS 5

2.1.1. Introducció a la generació procedimental de continguts 5

2.1.1.1. Jocs que utilitzen GPC 5

2.1.1.2. Taxonomia de la GPC 6

2.1.1.3. Què es pot generar de forma procedimental en un videojoc? 9

2.1.1.4. Propietats desitjables en un algorisme de generació procedimental 12

2.1.2. Tipus de generació i algorismes 14

2.1.2.1. Generadors de nombres pseudo-aleatoris 14

2.1.2.2. Gramàtiques generatives 14

2.1.2.3. Filtratge d'imatges 14

2.1.2.4. Algorismes espacials 16

2.1.2.5. Modelatge i simulació de sistemes complexos 19

2.1.2.6. Intel·ligència artificial 20

2.1.3. Representació de coneixements 23

2.1.3.1. Fragments experiencials 23

2.1.3.2. Plantilles 24

2.1.3.3. Components 24

2.1.3.4. Subcomponents 24

2.1.3.5. Mescla i combinació 25

2.2. GRAMÀTIQUES EN LA GPC 26

2.2.1. Què son les gramàtiques? 26

2.2.1.1. Gramàtiques formals 26

2.2.1.2. La jerarquia de Chomsky 27

2.2.2. Evolució de les gramàtiques i tipus d'algorismes. 30

2.2.2.1 Sistemes-L 30

2.2.2.2. Gramàtiques de forma 32

2.2.2.3. Gramàtiques de conjunts 35

2.2.2.4. Gramàtiques de divisió 36

2.2.2.5. Gramàtiques de parets. 39

2.2.2.6. Gramàtiques de forma CGA 40

2.2.2.7. Nodes 43

3. REFERENTS 45

3.1. TOWNSCAPER 45

3.2. THE ARCHITECT: PARIS 50

3.3. EINES DE UNITY 54

3.3.1. Procedural Generation Grid 54

3.3.2. ProGen 55

3.3.3. Building Crafter 56

4. OBJECTIUS 57

5. DISSENY METODOLÒGIC I CRONOGRAMA 58

5.1. METODOLOGIA 58

5.2. CRONOGRAMA 60

6. DESENVOLUPAMENT 61

6.1. CARACTERÍSTIQUES DE L'EINA 61

6.2. GRAMÀTICA GENERATIVA 62

6.2.1. Formes 62

6.2.2. Procés de producció o derivació 62

6.2.2. Normes de producció 63

6.2.2.2. Normes de tipus Àmbit (scope) 63

6.2.2.3. Normes de tipus Divisió (split) 64

6.2.2.4. Normes de tipus Repetició (repeat) 65

6.2.2.5. Normes de tipus Quadrícula (grid) 66

6.2.2.6. Normes de tipus Marge (margin) 67

6.2.2.7. Normes de tipus Component (component) 68

6.2.3. Post-producció 69

6.3. IMPLEMENTACIÓ DE LA GRAMÀTICA A UNITY 70

6.3.1. Formes 72

6.3.2. Normes de producció 72

6.3.3. Normes de post-producció 72

6.3.4. Gramàtica 73

6.3.5. Derivació 73

6.3.6. Post-producció 74

6.3.5. Ordre de fitxers 75

6.3.6. Interfícies 75

6.3.6.1. Editor de gramàtiques 75

6.3.6.2. Constructor de formes terminals 78

6.3.6.3. Constructor 79

7. RESULTATS 80

7.1. MANSIÓ DETERIORADA 81

7.2. PAGODA 84

7.3. LÍNIES DE MILLORA 87

7.3.1. Variables en els paràmetres. 87

7.3.2. Capes de derivació. 87

7.3.3. Snapping i Oclusió. 87

7.3.4. Millores en l'editor de gramàtiques 87

7.3.5. Millora de la norma d'àmbit 88

7.4. NOM DE LA EINA 88

8. CONCLUSIÓ 89

9. REFERENCIES 91

9.1. PROGRAMARI 95

10. ANNEX

GBBG DOCUMENTATION

Índex de figures

FIGURA 2-1. CONTINGUTS QUE ES PODEN GENERAR DE FORMA PROCEDIMENTAL EN VIDEOJOCS. 9

FIGURA 2-2. EXEMPLE DE LA DETECCIÓ DE VORES UTILITZANT MORFOLOGIA BINARIA. 15

FIGURA 2-3. EXEMPLE DE CONVOLUCIÓ EN UNA MATRIU BIDIMENSIONAL. 16

FIGURA 2-4. EXEMPLE D'ENRAJOLAT. 17

FIGURA 2-5. EXEMPLE DE ESTRATIFICACIÓ.. 17

FIGURA 2-6. FRACTAL DE KOCH. 18

FIGURA 2-7. DIAGRAMA DE VORONOI. 19

FIGURA 2-8. EL JOC DE LA VIDA DE CONWAY, UN DELS AUTÒMATS CEL·LULARS MÉS CONEGUTS. 19

FIGURA 2-9. RESULTATS DE CHEN ET AL. (2008).. 20

FIGURA 2-10. NORMES BÀSIQUES D'UN ALGORISME GENÈTIC PER CREAR LA SEGÜENT GENERACIÓ. 21

FIGURA 2-11. ESQUEMA DE L'ESTRUCTURA D'UNA XARXA NEURONAL ARTIFICIAL. 23

FIGURA 2-12. LES RESTRICCIONS DE LA JERARQUIA DE CHOMSKY. 27

FIGURA 2-13. UNA FSG, REPRESENTADA COM A NORMES DE REESCRIPTURA (A) I GRÀFICAMENT (B). 29

FIGURA 2-14. EXEMPLE D'UNA DERIVACIÓ EN UN SISTEMA-D0L. 30

FIGURA 2-15. EXEMPLE D'ARBRE FRACTAL. 31

FIGURA 2-16. CAPTURA DE PANTALLA DE ARCGIS CITYENGINE (2008). 32

FIGURA 2-17. EXEMPLE DE LA DEFINICIÓ D'UNA GRAMÀTICA DE FORMA. 34

FIGURA 2-18. EXEMPLE DE DERIVACIÓ D'UNA GRAMÀTICA DE FORMES. 35

FIGURA 2-19. LES NORMES D'UNA GRAMÀTICA DE DIVISIÓ SIMPLE. 36

FIGURA 2-20. RESULTAT DE LA DERIVACIÓ DE LA GRAMÀTICA DEFINIDA A LA FIGURA 2-19. 37

FIGURA 2-21. ESQUEMA DE LA INTERACCIONS ENTRE ELS DIFERENTS ELEMENTS QUE CONFORMEN EL *FRAMEWORK*... 37

FIGURA 2-22. EXEMPLE DE DERIVACIÓ D'UN EDIFICI AMB GRAMÀTIQUES DE PARET 40

FIGURA 2-23. EXEMPLE D'EDIFICIS GENERATS AMB GRAMÀTIQUES DE FORMA CGA. 41

FIGURA 2-24. A LA ESQUERRA UNA CONFIGURACIÓ. A LA DRETA UNA APROXIMACIÓ DE LES CARES OCLOSES. 42

FIGURA 2-25. ESQUERRA: EDIFICI GENERAT SENSE OCLUSIÓ NI *SNAPPING*. DRETA: EDIFICI GENERAT AMB OCLUSIÓ I *SNAPPING*. 43

FIGURA 2-26. RESULTATS DE (PATOW, 2012). 44

FIGURA 3-1. CAPTURA DE PANTALLA DEL VIDEOJOC TOWNSCAPER. 45

FIGURA 3-2. LES 5 FASES, D'ESQUERRA A DRETA, DE LA GENERACIÓ DE LA QUADRICULA. 46

FIGURA 3-3. LES POSSIBLES TIPOLOGIES DE LES CEL·LES EN L'ALGORISME. 47

FIGURA 3-4. EXEMPLE DE L'ALGORISME ORIGINAL. 48

FIGURA 3-5. CAPTURA DE PANTALLA DEL VIDEOJOC *THE ARCHITECT: PARIS* (2021). 50

FIGURA 3-6. EXEMPLES D'ESTILS DE FAÇANES. 51
FIGURA 3-7. EXEMPLE D'EXTRUSIONS DE FAÇANA. 52
FIGURA 3-8. EXEMPLES D'ESPAIS GENERATS UTILITZANT *PROCEDURAL GENERATION GRID*. 54
FIGURA 3-9. EXEMPLE D'EDIFICIS GENERATS UTILITZANT *PROGEN*. 55
FIGURA 3-10. EXEMPLE D'EDIFICIS GENERATS UTILITZANT *BUILDING CRAFTER*. 56
FIGURA 5-1. LA METODOLOGIA SCRUM. 59
FIGURA 5-2. CRONOGRAMA DEL PROJECTE. 60
FIGURA 6-1. EXEMPLE DE NORMA DE TIPUS ÀMBIT.. 64
FIGURA 6-2. EXEMPLE DE LA APLICACIÓ DE LA NORMA DE TIPUS DIVISIÓ.. 65
FIGURA 6-3. EXEMPLE DE L'APLICACIÓ D'UNA NORMA DE REPETICIÓ. 66
FIGURA 6-4. EXEMPLE DE L'APLICACIÓ D'UNA NORMA DE QUADRÍCULA. 67
FIGURA 6-5. EXEMPLE DE L'APLICACIÓ D'UNA NORMA DE MARGE. 68
FIGURA 6-6. EXEMPLE DE LA APLICACIÓ D'UNA NORMA DE COMPONENT 69
FIGURA 6-7 DIAGRAMA DE CLASSES DEL FRAMEWORK. 71
FIGURA 6-8. IMPLEMENTACIÓ DE L'ALGORISME DE DERIVACIÓ 74
FIGURA 6-9. INTERFÍCIE DE L'EDITOR DE GRAMÀTIQUES. 76
FIGURA 6-10. COMPARACIÓ D'INSPECTORS. 77
FIGURA 6-11. EXEMPLE DE MISSATGE D'AJUDA 78
FIGURA 6-12. INTERFÍCIE DEL CONSTRUCTOR DE FORMES TERMINALS. 78
FIGURA 6-13. INTERFÍCIE DEL CONSTRUCTOR. 79
FIGURA 7-1. EXEMPLE DE GENERACIÓ UTILITZANT LA GRAMÀTICA "MANSIÓ DETERIORADA". 81
FIGURA 7-2. NORMES DE LA GRAMÀTICA DE LA MANSIÓ. 82
FIGURA 7-3. PROCÉS DE DERIVACIÓ.. 82
FIGURA 7-4. POST-PRODUCCIÓ DE L'EXEMPLE DE LA FIGURA 7-3. 83
FIGURA 7-5. EXEMPLE DE GENERACIÓ UTILITZANT LA GRAMÀTICA "PAGODA". 84
FIGURA 7-6. NORMES DE PRODUCCIÓ DE LA GRAMÀTICA "PAGODA" 85
FIGURA 7-7. ELS PASSOS DE LA GENERACIÓ D'ESQUERRA A DRETA. 86

0. Glossari de termes

- **Algorisme**: Un conjunt finit d'instruccions o passos que serveixen per a executar una tasca o resoldre un problema
- ***Asset***: Terme usat de forma comú en la industria per referir-se als fragments que formen un videojoc. (veure 1.1.3.1 del marc teòric)
- **Autòmat linealment acotat:** És una màquina de Turing no determinista en la que en lloc de tenir una cinta potencialment infinita sobre la qual calcular, el càlcul es restringeix a la porció de la cinta que conté l'entrada més els dos quadrats de cinta que subjecten els marcadors finals.
- **Autòmat d'estats finits**: És un model matemàtic d'un sistema compost per estats, transicions i accions. Un estat emmagatzema informació del passat. Una transició indica un canvi d'estat i es descriu per la condició que és necessària acomplir per activar la transició. Una acció és una descripció d'una activitat que es realitza en un moment donat. D'accions n'hi ha de diversos tipus:
    - Acció d'entrada: executa l'acció quan s'entra a l'estat.
    - Acció de sortida: executa l'acció quan s'abandona l'estat.
    - Acció d'Input: executa l'acció depenent de l'estat actual i les condicions d'entrada.
    - Acció de Transició: executa l'acció quan succeeix una certa transició.
- **Autòmat amb pila:** Un autòmat d'estats finits que treballa sobre una pila. Amb aquesta pila pot: utilitzar el símbol dalt de tot de la pila per decidir el següent estat, manipular la pila afegint o retirant símbols.
- **Axioma:** Conjunt de símbols inicials en una derivació.
- ***Bitmap*:** representació d'una imatge utilitzant una sèrie de columnes i files de píxels, cada píxel guarda un valor de un escalar (blanc i negre) o un vector (color i transparència).
- ***City-builder*:** Gènere de videojocs que s'enfoca en la simulació de construcció de ciutats.
- **Derivació:** Procés que consisteix en aplicar normes de producció al axioma fins a arribar al resultat final.

- **Escena:** Espai virtual on el motor de jocs col·loca tots els elements visibles d'un videojoc: Imatges, models 3D, càmera, il·luminació, etcètera.
- ***Framework***: La estructura bàsica d'un sistema més complex. En l'àmbit de la programació un *framework* és una eina que proporciona components ja fets o solucions personalitzades per accelerar el desenvolupament.
- **GameObject:** En el motor Unity s'anomena *GameObject* tots els objectes que formen part de la escena.
- ***Gameplay***: La forma característica en què es produeix o s'experimenta l'acció d'un videojoc.
- **Llavor**: Un numero o vector utilitzar per inicialitzar un generador de nombres pseudo-aleatoris
- **Màquina de Turing:** És un model matemàtic que consisteix en un autòmat capaç d'implementar qualsevol problema matemàtic expressat per mitjà d'un algoritme. L'autòmat treballa sobre una cinta dividida en cel·les, cada una d'elles pot guardar un símbol d'un conjunt finit. Aquest autòmat segueix un seguit de normes les qual el poden dur a moure's a dreta o esquerra, sobreescriure el símbol de la cel·la, o aturar-se.
- **Material:** Estructura de dades que conté la informació per definir com es veu un model 3D quan és renderitzat.
- **Model generador / Model de generació / Generador**: Un algorisme que genera continguts.
- **Normes de producció:** normes que especifiquen quan i com es substitueix un símbol per un altre.
- ***Offline***: Codi que s'executa una vegada per generar un contingut, i aquest s'utilitza en el joc.
- **Online**: Codi que s'executa en temps real juntament amb el videojoc.
- **Prefab:** Són plantilles que defineixen *gameObjects* en el motor Unity, permeten instanciar copies d'aquesta plantilla fàcilment.
- ***Quads*:** Quadrilàters que formen la malla d'un model 3D, cada quad es divideix en dos triangles per a renderitzar la imatge, però és més senzill treballar amb quads a l'hora de modificar un model 3D.

- **Rejugabilitat**: Terme utilitzat en l'àmbit dels videojocs i els jocs de taula per descriure el potencial que té un títol de seguir sent jugat després d'acabar el joc per primera vegada.
- **Roguelike i Roguelite**: Gèneres de videojocs inspirats en el videojocs *Rogue* (Ai Design, 1980). El primer manté de forma estricta les mecàniques principals mentre que els segon compren tot tipus de variacions.
- **Singleton:** Estructura d'una classe per permetre que aquesta sigui utilitzada de forma estàtica sense que realment ho sigui.
- ***Stencil buffer*:** És un buffer de dades extra que s'utilitza en hardware de gràfics modern. Aquest buffer és utilitzat normalment per crear màscares per definir una plantilla de quines parts de la imatge s'han de renderitzar.
- ***Tileset*:** conjunt de tipus de peces que formen un enrajolat. (veure 1.2.4.1 del marc teòric)
- **Topologia:** Les propietats espacials d'un objecte.
- **Unity**: Un motor de videojocs multi-plataforma. És el motor més utilitzat en la industria i l'elegit per dur a terme aquest treball.
- **Videojocs AAA**: videojocs de gran producció, creats per un equip molt nombrós i amb un gran pressupost.
- **Vocabulari:** Nombre finit de símbols que formen un llenguatge. En el cas d'un llenguatge formal poden ser símbols terminals o símbols no terminals.

1. Introducció

Des de l'inici dels videojocs, s'ha vist que existeix una tendència a cada vegada superar el que s'ha fet anteriorment. A la punta de llança de la industria, sempre s'ha intentat portar els videojocs al extrem al qual el hardware ens ofereix, sent cada vegada més ambiciosos amb el que el videojoc pot oferir, sobretot des de la dècada dels 90, on els gràfics 3D es van introduir al gran públic. Això va comportar el desplaçament de l'esforç, reduint el pes de la programació i augmentant poc a poc la quantitat de treball dedicada a crear el món de joc. Aquesta tendència ha portat a que cada vegada els videojocs AAA s'enfoquin a mapes més grans, a món oberts, i que ho facin sense sacrificar cap mena de detall. És per això que veiem desenvolupaments de videojocs AAA, que tot i tenir una gran plantilla de treballadors dedicant-s'hi, tenen uns desenvolupaments llargs, de més de 5 anys.

Per reduir l'esforç que requereix la creació de milers de *assets* i la col·locació d'aquests en el món de joc, desenvolupadors de tot el món han anat construint durant anys un seguit de tècniques i algorismes per automatitzar aquest procés. Aquest conjunt de tècniques és conegut com a Generació Procedimental de Continguts (GPC).

Tot i que els mètodes de GPC serveixen per generar tot tipus d'elements que formen un joc, aquest treball es centra en els models generatius basats en gramàtiques, en concret en exteriors d'edificis. Per arribar al resultat desitjat s'analitzaran diferents videojocs comercials que incorporen contingut generat de forma procedimental, així com eines que permeten la generació d'aquest i l'estudi de les tècniques i mètodes mes rellevants en el camp de la GPC. Finalment s'aplicaran els coneixements recopilats per desenvolupar un *framework* que permeti la creació d'edificis detallats amb la mínima interacció de l'usuari, tot i permetent que aquest modifiqui les característiques generals.

Aquest projecte es desenvoluparà utilitzant Unity, un dels motors més populars en la industria del videojoc i per el qual existeix una gran quantitat de documentació i ajuda, a més a més, de ser el motor amb el qual s'han desenvolupat la majoria de referents del projecte.

2. Marc teòric

2.1. Generació procedimental de continguts

2.1.1. Introducció a la generació procedimental de continguts

Un dels principals punts d'aquest treball és la generació procedimental de continguts (GPC), per tant abans de començar, és convenient explicar el que significa. Segons el Diccionari de l'Institut d'Estudis Catalans (DIEC2), "generació" en aquest context és "la acció de generar (crear, produir)", "procedimental" significa "Relatiu o pertanyent a un procediment (manera de procedir, d'obrar, per aconseguir un resultat)" i finalment "contingut" significa "Cosa continguda dins una altra", que en el cas dels videojocs pot incloure: mons, textures, diàlegs, missions, etc. Pràcticament totes les peces que formen part d'un videojoc. Per tant podem definir-la semànticament com a la creació o producció mitjançant un procediment (en aquest cas automàtic) de una o més parts que formen un videojoc.

La GPC en l'àmbit dels videojocs ha estat definida de diferents maneres, com ara: la creació de continguts automàticament utilitzant algorismes i amb entrades de l'usuari limitades o indirectes (Togelius, Kastbjerg, Schedl, & Yannakakis, 2011) o l'ús d'algorismes per produir continguts de jocs que normalment serien produïts per un dissenyador, com ara textures, efectes de so, mapes, nivells, personatges, armes, missions, o fins i tot mecàniques i normes (Barriga, 2019).

Un concepte que apareix repetidament en les definicions acadèmiques de la GPC son els algorismes, un algorisme és segons el DIEC2: "El conjunt de regles per a resoldre un problema en un nombre finit de passos". En aquest cas, el problema a resoldre és la generació de continguts amb una intervenció manual nul·la o quasi nul·la.

2.1.1.1. Jocs que utilitzen GPC

L'ús de la GPC en videojocs té una llarga història, les motivacions més populars dels pioners que van començar a utilitzar la GPC en videojocs eren la rejugabilitat i la adaptabilitat (Smith, Procedural Content Generation: An Overview, 2015). La GPC pot proveir al jugador grans quantitats de continguts de manera que cada vegada que torni a jugar l'experiència sigui diferent, per altra banda, es pot aprofitar que els continguts del joc no estan determinats al

cent per cent per tal de generar continguts tenint en compte el nivell d'habilitat del jugador per tal d'adaptar-se el millor possible al jugador en particular, en comptes del jugador mitjà.

Als inicis la GPC es va utilitzar per optimitzar la memòria que ocupaven els videojocs, aquest és el cas de *Elite* (Acornsoft, 1984), que es generava de forma procedimental un univers expansiu a partir de una llavor (seed). Tot i així la GPC era completament determinista, ja que la llavor estava seleccionada prèviament. En altres videojocs també es va utilitzar per a la generació de nivells, com és el cas de *Rogue* (AI Design, 1980), on les masmorres es generen de manera que no en surtin mai dues d'iguals, i que va donar nom al gèneres *roguelike* i més endavant *roguelite*. També trobem exemples d'ús de la GPC en la generació d'escenaris, com ara *Civilization* (MicroProse, 1991) o *Minecraft* (Mojang, 2011) que creen l'escenari al complet perquè sigui diferent cada partida.

Més cap a l'actualitat trobem exemples de generació procedimental aplicada a altres aspectes del joc, com ara la generació d'armes de *Borderlands* (Gearbox, 2009) o la generació de criatures a *Spore* (Maxis, 2008). Finalment, l'últim exemple que vull posar sobre la taula és la GPC portada a l'extrem per part del videojoc *Slaves to Armok: God of Blood Chapter II: Dwarf Fortress* (Bay 12 Games, 2006), que pràcticament tot el contingut es genera de forma procedimental: mon, ítems, civilitzacions, religions, llegendes, etc.

2.1.1.2. Taxonomia de la GPC

Per tal de categoritzar les propietats que té un mètode de GPC, Togelius, Yannakakis, Stanley i Browne (2010) presenten i després Shaker, Togelius i Nelson (2016, p. 1-15) expandeixen una estructura que permet identificar les diferències i similituds entre aproximacions a la GPC. Consisteix en 7 dimensions, on normalment s'hauria de pensar que un mètode o una solució individual es troba en algun punt d'un continu entre els extrems d'aquesta dimensió.

2.1.1.2.1. Online versus *offline*

A l'hora de generar continguts mitjançant GPC aquest contingut pot ser generat de manera online, el contingut es genera mentre el jugador està jugant, això permet que existeixi una gran quantitat de variacions que poden permetre una jugabilitat infinita i també obren la porta al contingut adaptatiu, que varia segons les accions del jugador. Un exemple clar de GPC online la trobaríem en jocs *endless runner* com ara *Temple Run* (Imangi studios, 2011), però també

trobem exemples en altres gèneres, per exemple, *Left 4 Dead* (Valve, 2008) altera la experiència del jugador en base a les seves accions, utilitzant tècniques de GPC per decidir la posició i freqüència dels enemics que apareixen per tal de mantenir els jugadors enfocats en el joc però sense aclaparar-los (Booth, 2009).

Per altra banda, també es pot utilitzar la GPC de manera *offline*, és a dir, el contingut es crea durant el desenvolupament o abans de iniciar la partida. Per exemple, *Forza Motorsport* (Turn 10 Studios, 2005) entrena de manera *offline* als personatges no jugables (NPCs) de manera que imitin el comportament dels jugadors (Shaker, Togelius, & Nelson, 2016, p. 8).

2.1.1.2.2. Necessari versus opcional

La GPC pot ser utilitzada per generar continguts que son necessaris per poder completar els nivells, o pot utilitzar-se per generar continguts auxiliars que poden ser substituïts. La principal diferència és que en el contingut necessari és imperatiu que es generi correctament, ja que una mala generació pot fer que un nivell no es pugui completar, per altra banda els continguts auxiliars tenen més flexibilitat. Les armes de Borderlands (Gearbox, 2009) son un exemple de contingut auxiliar, ja que és necessari una generació molt precisa per poder completar els nivells. En quant a continguts necessaris un bon exemple és la generació de nivells de *roguelites* com ara *Minecraft Dungeons* (Mojang, 2020).

2.1.1.2.3. Grau i dimensions de control

La generació de continguts pot ser controlada de diverses formes. Les més utilitzades són l'ús de llavors aleatòries, que atorga molt control sobre l'espai de generació; o bé l'ús de paràmetres per controlar els continguts generats al llarg d'una sèrie de dimensions.

L'exemple més conegut de l'ús de llavors aleatòries es *Minecraft* (Mojang, 2011), que permet que el món es generi de la mateixa manera sempre si s'utilitza la mateixa llavor.

2.1.1.2.4. Genèric versus adaptatiu

El contingut adaptatiu fa referència al paradigma de la GPC on les accions del jugador afecten directa o indirectament al contingut generat, per tant aquest està personalitzat i/o centrat en el jugador. Per altra banda, el contingut genèric és el que no està influenciat per els comportaments del jugador. La majoria de videojocs comercials aproximen la GPC d'una manera genèrica, tot i així hi ha varis títols que utilitzen la GPC adaptativa: Left 4 Dead (Valve,

2008) com s'ha explicat anteriorment utilitza la GPC adaptativa per ajustar-se automàticament al ritme del jugador, modificant la dificultat i per tant mantenir el jugador atent.

2.1.1.2.5. Estocacitat versus determinisme

La GPC determinista permet la regeneració d'un mateix contingut si es parteix des del mateix punt d'inici. Contràriament, la GPC estocàstica no permet la regeneració de resultats.

Minecraft (Mojang, 2011) és un clar exemple de GPC determinista ja que sempre es pot regenerar un món a partir de la llavor.

2.1.1.2.6. Constructiu versus generar-i-provar

En la GPC constructiva el contingut és generat directament, a diferència de la GPC de generar-i-provar, on es genera contingut de forma recursiva fins a trobar una solució satisfactòria. El mètode constructiu el podem trobar en la generació de nivells del gènere *roguelike* i *roguelite*. Per altra banda, *Yavalath* (Browne, 2007) és un joc de taula per a dos jugadors generat completament per un ordinador mitjançant el mètode de generar-i-provar (Browne & Marie, 2010).

2.1.1.2.7. Generació automàtica versus autoria mixta

La generació automàtica és el paradigma dominant a la GPC ja que l'autoria mixta és bastant recent en comparació. La generació automàtica permet només una interacció limitada amb l'algorisme de GPC a través de paràmetres per controlar el contingut que es generat. En canvi, el paradigma d'autoria mixta permet al dissenyador una interacció molt més profunda amb el model de GPC.

En el procés d'autoria mixta, el dissenyador proporciona una part del contingut a generar i el model de GPC l'expandeix tenint en compte l'input del dissenyador. Existeixen varis exemples de sistemes que fan ús d'aquest paradigma, com ara *Tanagra* (Smith, Whitehead, & Mateas, 2010), una eina on el dissenyador dibuixa una part d'un nivell 2D i un algorisme genera les parts que falten tenint en compte unes limitacions per conservar la jugabilitat. Un altre exemple és *SketchaWorld* (Smelik, Tutenel, Kraker, & Bidarra, 2010) una eina que a partir de simples *sketches* en CAD (punts i línies), permet generar de forma procedimental mons sencers.

2.1.1.3. Què es pot generar de forma procedimental en un videojoc?

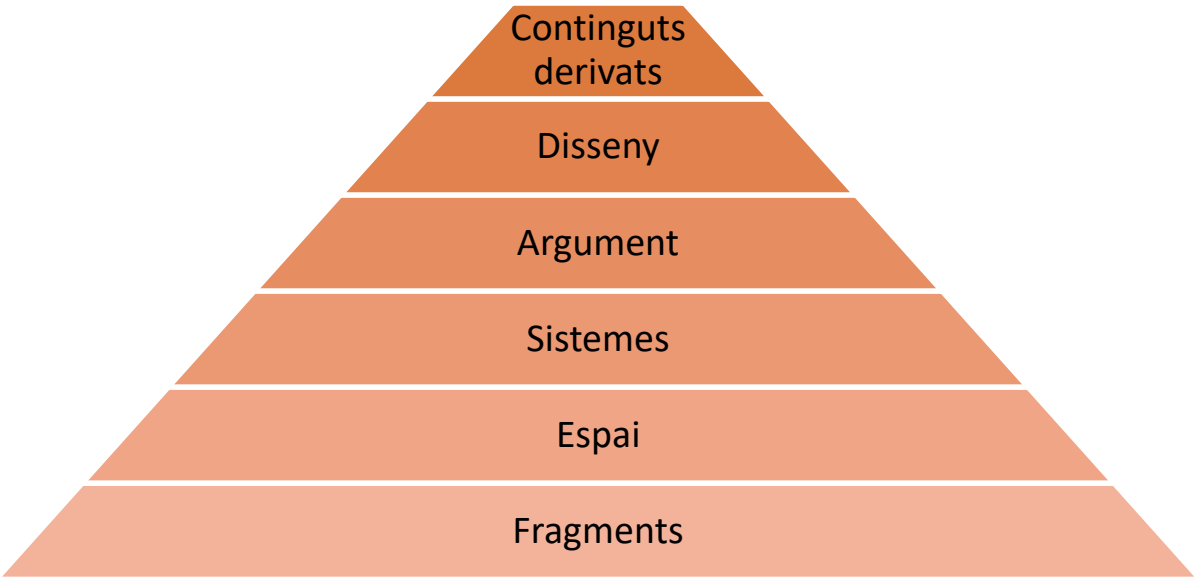


Figura 2-1. Continguts que es poden generar de forma procedimental en videojocs (Hendrikx, Meijer, Van Der Velden, & Iosup, 2013).

La GPC es pot utilitzar per generar una gran quantitat de aspectes diferents en un videojoc, per classificar els tipus de continguts que poden generar-se Hendrikx, Meijer, Van Der Velden i Iosup (2013) creen una classificació on divideixen el contingut que es pot generar de forma procedimental en sis categories. Aquestes categories s'estructuren en forma de piràmide, on les categories situades més amunt acostumen a ser construïdes a partir dels elements dels nivells més baixos.

2.1.1.3.1. Fragments

Els fragments son les peces més elementals d'un contingut, per ells mateixos no tenen gaire significat. Podem trobar cinc tipus de fragments:

- **Textures.** Les textures son imatges que donen la representació visual dels elements del joc, ja sigui en models 3D per donar-los-hi detall, en menús, o en elements d'un videojoc 2D. Les textures acostumen a definir l'estil visual del videojoc.
- **Sons.** El so és una peça clau en els videojocs (Bartle, 2003), la música determina la atmosfera i el ritme del videojoc i els efectes de so donen feedback al jugador sobre les seves accions i els canvis en l'entorn.

- **Models 3D.** Els models els trobem únicament en videojocs 3D, ja que el seu equivalent 2D son textures. Els models donen la forma a l'escenari i influeixen directament en el *gameplay*.
- **Comportaments.** Els comportaments són la manera en la que els objectes interactuen amb el seu entorn. Els comportaments procedimentals s'utilitzen normalment per crear una il·lusió de complexitat.
- **Partícules i efectes.** Les partícules i els efectes (foc, aigua, núvols, etcètera) s'utilitzen per donar realisme al món de joc. Al principi el seu rol era purament decoratiu però amb el temps s'ha anat augmentant el detall, el realisme i la interactivitat.

2.1.1.3.2. Espais

L'espai de joc és l'entorn on passa l'acció, i està format per fragments entre els quals els jugadors es mouen. L'espai de joc té un paper fonamental en la interpretació del joc que fan els jugadors (Nitsche, 2008). Hendrikx et al. (2013) divideixen els espais en dues categories: interior i exterior. Aquesta distinció es fa per varis motius. Per motius estètics, ja que els espais exteriors acostumen a tenir temàtiques similars, mentre que els espais interiors poden variar en gran manera gràcies a que estan majoritàriament dividits en sales. Per motius culturals, ja que els espais interiors son més intricats i acostumen a requerir més *assets* úniques. Finalment, per motius tècnics, ja que requereixen aproximacions diferents al generar-los de forma procedimental.

2.1.1.3.3. Sistemes

L'ús de sistemes complexos per generar o simular parts del videojoc és una pràctica bastant comú. L'ús d'aquests sistemes aporta als videojocs una capa de realisme i versemblança extres. Els tipus de sistemes que Hendrikx et al.(2013) proposen son:

- **Ecosistemes.** Els ecosistemes determinen la localització, evolució i interacció de la flora i fauna mitjançant algorismes i regles. Un exemple de ecosistema el trobem a *Ultima Online* (Origin Systems, 1997), on es generaven ecosistemes amb cadenes alimentaries complexes (Vieira, Dembogurski, Alvim, & Braida, 2018).
- **Espais urbans.** Els espais urbans estan formats per agrupacions d'edificis on molta gent hi viu, interactua amb l'entorn i fa evolucionar aquest espai durant anys. Per generar

espais urbans, els algorismes acostumen a allunyar-se d'aquest punt de vista i normalment es fa generant primer els carrers, després els solars i finalment els edificis.

- **Xarxes de camins / carreteres.** Les xarxes de carreteres son les estructures bàsiques dels espais exteriors. Serveixen per el transport entre punts d'interès i estructuren l'espai. El principal obstacle a la hora de generar una xarxa es trobar el balanç entre l'estructura i la aleatorietat.
- **Comportaments d'entitats.** En interaccions complexes entre el jugador i l'entorn serveixen per donar-li vida al món de joc i els *NPCs* son una peça fonamental per aconseguir-ho.

2.1.1.3.4. Argument o guió

L'argument descriu, sovint de manera transparent per a l'usuari, la manera i l'ordre en què es desenvolupen els esdeveniments del joc.

- **Puzles.** Els puzles son obstacles o problemes al qual el jugador els hi ha de trobar una solució basant-se en la informació que ha obtingut o explorant l'espai per trobar possibles solucions (Colton, 2002). Per els puzles el procés de trobar la solució és el joc en si. La dificultat dels puzles està determinada per l'espai de solucions i per l'experiència prèvia el jugador.
- ***Storyboards*.** Els *storyboards* son ajudes per el dissenyador, normalment es presenten en format de còmic, una seqüència de panells que descriuen els esdeveniments de l'escena. Els *storyboards* poden ser utilitzats també per entretenir el jugador, per exemple, cinemàtiques entre parts del *gameplay*.
- **Història.** La història acostuma a ser una de les parts més importants d'un videojoc, ja que manté el jugador motivat, defineix quins son els esdeveniments més importants del joc i proporciona al jugador l'objectiu principal per aconseguir.
- **Nivells.** El concepte de nivell s'utilitza en la majoria de videojocs com a separador de seqüències de *gameplay*.

2.1.1.3.5. Dissenys

El disseny d'un joc està determinat per un seguit de normes (què es pot fer en el joc?) i objectius (què intenta aconseguir el jugador?). També tenen importància els elements com

ara un arc narratiu o un tema gràfic. (Norman, 2002). Per tant, un joc es pot descriure com a una instancia concreta d'un disseny de joc, on hi ha uns paràmetres que s'han definit.

- **Disseny de sistemes.** El disseny de sistemes és "la creació de patrons matemàtics subjacents al joc i les regles del joc" (Brathwaite & Schreiber, 2008).
- **Disseny de món.** El disseny de món està format per "el disseny d'un escenari, una història i una temàtica" (Brathwaite & Schreiber, 2008).

2.1.1.3.6. Continguts derivats

Els continguts derivats son els continguts que es creen a partir del món de joc. Els continguts derivats augmenten la immersió del jugador en el món de joc, ja que els jugadors graven les seves experiències del joc per mostrar-les a altres jugadors. Aquest "joc més enllà del joc" és la base del concepte de "metajoc" (Garfield, 2000).

- **Notícies i emissions.** Els desenvolupadors poden mostrar els nous canvis en el món de joc. Els jugadors poden compartir la seva sessió de joc, ja sigui en vídeo o en directe.
- **Taules de classificació**. Classificacions de jugadors.
- **Fòrums i *wikis*:** Pàgines web on el contingut es aportat per els jugadors, ja sigui responent dubtes concrets, debats sobre el videojoc o recopilació de les característiques del videojoc.

2.1.1.4. Propietats desitjables en un algorisme de generació procedimental

Una implementació d'un mètode de GPC és la solució a un problema de generació. Per poder avaluar si un algorisme és o no una bona solució es pot fer en base a un seguit de propietats que hauria de tenir (Shaker, Togelius, & Nelson, 2016, p. 6):

- **Velocitat.** Els requeriments d'un model poden variar àmpliament, des de mili-segons fins a mesos, depenent si el procés de generació és online o *offline*. En el cas que sigui online, el model no pot trigar gaire més d'uns mili-segons a generar el contingut.
- **Fiabilitat.** Cada problema necessita un nivell de fiabilitat diferent i l'algorisme ha de complir amb aquest nivell de fiabilitat. Per exemple, si una masmorra es genera sense la porta de sortida és un problema catastròfic, però si una flor es genera torta no suposa cap problema.

- **Control.** Els algorismes de GPC necessiten que es puguin controlar d'alguna manera, ja sigui l'usuari o un algorisme extern (p. e., un algorisme de adaptació al jugador).
- **Expressivitat i diversitat.** Per evitar que tots els resultats d'un algorisme siguin massa similars, aquest necessita diversitat, però no massa. Un extrem inexpressiu seria un algorisme que sempre té com a resultat el mateix nivell però canvia el color d'una pedra i a l'altre extrem d'expressivitat seria un algorisme que ajunta components d'una manera caòtica.
- **Creativitat i versemblança.** En la majoria de casos, el resultat no ha de semblar que s'hagi generat de forma procedimental. Cal que l'algorisme generi continguts que podrien ser creats per un ésser humà.

2.1.2. Tipus de generació i algorismes

Per resoldre el problema de la generació procedimental de qualsevol tipus de continguts podem abordar-lo des de molts angles diferents. Cada mètode és més efectiu per un tipus de continguts i per tant, per generar una gran varietat de continguts han aparegut una gran varietat de mètodes, que Hendrikx et al. (2013) divideixen en sis grans grups. Cada un d'aquests grups està compost per varis mètodes.

2.1.2.1. Generadors de nombres pseudo-aleatoris

Per aproximar-nos a la sensació de aleatorietat que ens dona la natura, com per exemple la varietat de terreny d'una muntanya, la forma d'una pedra o la forma dels núvols, s'utilitzen generadors de nombres pseudo-aleatoris o PRNGs per les seves sigles en anglès. Utilitzant PRNGs podem imitar el comportament erràtic que presenta sovint la natura. (Hendrikx, Meijer, Van Der Velden, & Iosup, 2013)

Els nombres resultants generats s'anomenen pseudo-aleatoris perquè no son aleatoris completament. Per aconseguir nombres completament aleatoris en ordinadors es necessita una entrada que depengui de fenòmens físics, per exemple, un vídeo en directe d'un entorn caòtic o un detector de partícules emeses per un material radioactiu. Com que la gran majoria de sistemes no tenen accés a aquest tipus de hardware s'utilitzen mètodes deterministes per generar nombres que semblin aleatoris. Tot i ser completament determinats aquests nombres tenen les propietats necessàries per simular els nombres aleatoris (Van Verth & Bishop, 2008).

2.1.2.2. Gramàtiques generatives

Les gramàtiques generatives deriven dels estudis lingüístics de Noam Chomsky durant la dècada de 1960, per tant podem esperar una similitud important amb el funcionament del llenguatge. Les gramàtiques generatives són un conjunt de normes que, operant en paraules individuals, poden generar només frases que son gramaticalment correctes (Hendrikx, Meijer, Van Der Velden, & Iosup, 2013). Aquestes frases podem utilitzar-les per crear objectes correctes a partir d'elements codificats com a lletres o paraules. (Veure apartat 2)

2.1.2.3. Filtratge d'imatges

El filtratge d'imatges té l'objectiu de millorar les imatges en un aspecte preestablert per tal de mesurar o emfatitzar característiques de la imatge que eren difícils de detectar prèviament

(Hendrikx, Meijer, Van Der Velden, & Iosup, 2013). Son moltes les tècniques de filtratge d'imatges que existeixen, les més rellevants per la generació procedimental són la morfologia binaria i els filtres de convolució.

2.1.2.3.1. Morfologia binaria

La morfologia binària és un conjunt de tècniques de processament d'imatges que treballen sobre imatges binàries, imatges en les que els píxels són blancs o negres. Aquestes imatges s'obtenen aplicant un llindar a una imatge original, de manera que tots els píxels per sota d'un determinat nivell d'intensitat es posen en negre i la resta en blanc. Les operacions bàsiques de morfologia binària són la dilatació i l'erosió, que impliquen afegir o eliminar píxels de la vora d'un objecte en una imatge. Combinant aquestes operacions bàsiques, es poden realitzar operacions més complexes, com ara detectar vores en una imatge, fent una dilatació i després restant-li la imatge original.

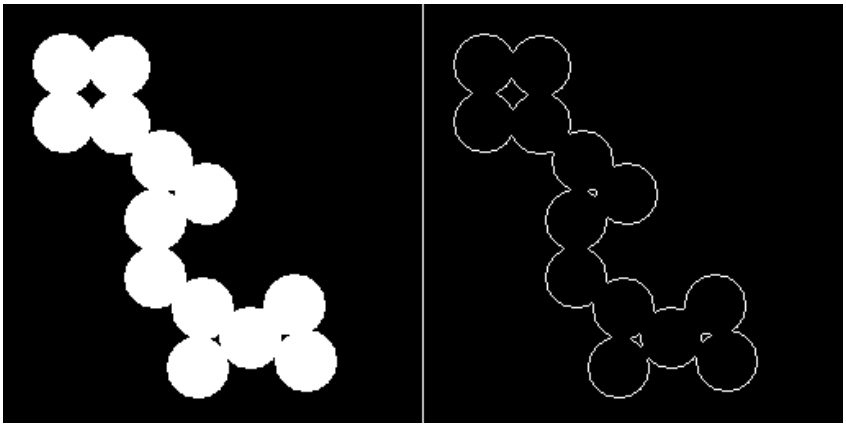


Figura 2-2. Exemple de la detecció de vores utilitzant morfologia binaria.

https://es.mathworks.com/help/images/morphological-dilation-and-erosion.html

2.1.2.3.2. Filtres de convolució

Els filtres de convolució, també anomenats *kernels*, son una petita imatge o matriu que actua com a filtre al executar una convolució entre el *kernel* i la imatge. La convolució és un operador matemàtic (representat amb el símbol $*$) que opera dos funcions o conjunts de dades, creant una nova funció o conjunt de dades que representa la superposició de les funcions o conjunts de dades originals. En el cas dels filtres d'imatges no s'opera amb funcions, sinó que es fa amb matrius que representen els píxels de la imatge.

Per exemple, si tenim dues matrius de tres per tres, la primera un *kernel* i la segona un fragment d'imatge, la convolució és el procés de recórrer tant les files com les columnes del *kernel* i multiplicar localment entrades similars i sumar. L'element a les coordenades [2, 2] (és a dir, l'element central) de la imatge resultant seria una combinació ponderada de totes les entrades de la matriu d'imatge, amb pesos donats pel nucli:

$$\left(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\right)[2,2] = (a \cdot 1) + (b \cdot 2) + (c \cdot 3) + (d \cdot 4) + (e \cdot 5) + (f \cdot 6) + (g \cdot 7) + (h \cdot 8) + (i \cdot 9)$$

Aplicant filtres de convolució es pot eliminar el soroll, difuminar, donar nitidesa, detectar vores o fins i tot el moviment d'un objecte (Hendrikx, Meijer, Van Der Velden, & Iosup, 2013). En els últims anys aquests filtres han guanyat notorietat per el seu ús en les Xarxes Neurals de Convolució (CNNs) per generar tot tipus d'imatges.

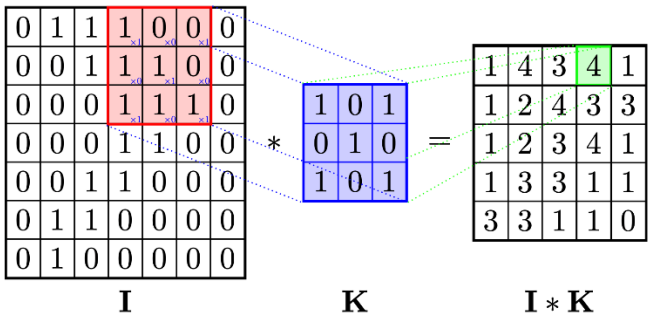


Figura 2-3. Exemple de convolució en una matriu bidimensional.

https://tex.stackexchange.com/questions/522118/visualizing-matrix-convolution

2.1.2.4. Algorismes espacials

En aquesta secció, tractarem diverses tècniques per generar contingut del joc mitjançant la manipulació de l'espai, com ara l'enrajolat i estratificació, la subdivisió de quadrícules, utilitzar fractals i l'ús de diagrames de Voronoi. Aquestes tècniques utilitzen inputs estructurats, com ara quadrícules o autorecurrència, per produir un output.

2.1.2.4.1. Enrajolat i estratificació

L'enrajolat o *tiling* (de l'anglès), és una tècnica que consisteix en dividir l'espai en una quadricula quadrada o hexagonal, de manera que el cobreixi l'espai uniformement. Aquestes quadrícules, tot i ser estructures de dades bidimensionals, es poden utilitzar per crear una il·lusió de 3D utilitzant perspectiva isomètrica.

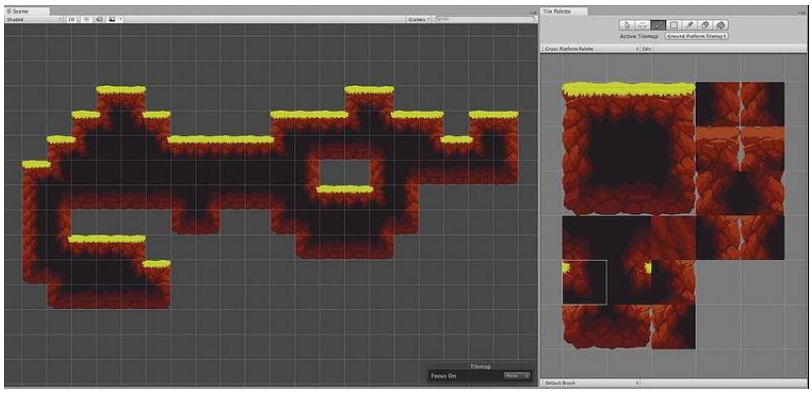


Figura 2-4. Exemple d'enrajolat. A la dreta, les possibles opcions per omplir cada casella. A l'esquerra un exemple de nivell.

L'estratificació o *layering* (de l'anglès) consisteix en agrupar en un sol nivell varies quadrícules enrajolades. Les quadrícules poden tenir parts transparents per deixar que es vegin les quadrícules que queden en una capa inferior. Aquesta tècnica permet la creació d'espais amb una sensació de profunditat només utilitzant una selecció limitada de textures.

Figura 2-5. Exemple de estratificació. Dividides amb la línia vermella hi ha dues capes diferents, una és amb la que interactua el jugador i l'altre es el fons.

2.1.2.4.2. Subdivisió de quadrícula

La subdivisió de quadrícula és un mètode per generar objectes mitjançant tècniques iteratives i dinàmiques. Comença dividint un objecte en una quadrícula uniforme amb textures adequades i després utilitza algorismes com l'algorisme Patch-LOD (Pi, Song, Zeng, & Li, 2006) per afegir detalls a l'objecte. L'aspecte dinàmic d'aquesta tècnica consisteix en ajustar el nivell de detall en funció de la perspectiva de l'espectador, només subdividir les cel·les de la quadrícula més properes a l'espectador per obtenir un nivell de detall més alt i deixar la resta de la quadrícula gruixuda per estalviar càlcul. Un exemple d'aquesta tècnica és la generació

de terreny procedimental, on només les cel·les properes al jugador estan ben detallades mentre que la resta del terreny és menys detallada.

2.1.2.4.3. Fractals

Els fractals són figures complexes que estan formades per còpies més petites de si mateixes, com ara el floc de neu Koch. Ofereixen un ampli ventall de possibilitats mitjançant la manipulació d'uns quants paràmetres, factor que els hi dona molta utilitat. Els fractals acostumen crear objectes que semblen tenir un detall infinit, però que es poden representar mitjançant una funció recursiva senzilla. El seu defecte és que generar-los pot ser computacionalment intensiu a causa de la seva naturalesa recursiva. Per millorar el rendiment, s'utilitzen sistemes de funcions iterades per generar la imatge mitjançant un procés iteratiu en lloc d'un recursiu.

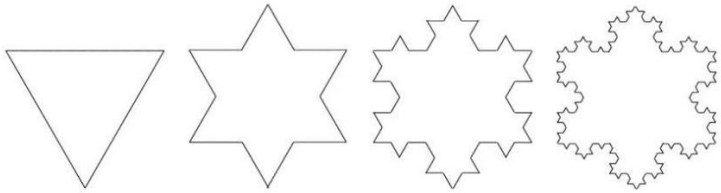


Figura 2-6. Fractal de Koch.

2.1.2.4.4. Diagrames de Voronoi

Els diagrames de Voronoi (Aurenhammer, 1991) son divisions de l'espai en que les peces en les que es divideix estan determinades per la posició d'un nombre indefinit de punts (punts d'interès) en el pla o l'espai. A partir d'aquests punts l'espai es divideix seguint la norma del veí més proper, és a dir, cada punt de l'espai s'associa al punt d'interès més proper. Les divisions de l'espai queden marcades per segments rectes que són equidistants dels punts d'interès als que separen.

De la mateixa manera que els fractals, al ser un càlcul iteratiu, a partir d'un cert nombre de punts d'interès és necessari utilitzar algorismes més eficients per calcular els diagrames, com per exemple els proposats per de Berg, Cheong, van Kreveld i Overmars (2008).

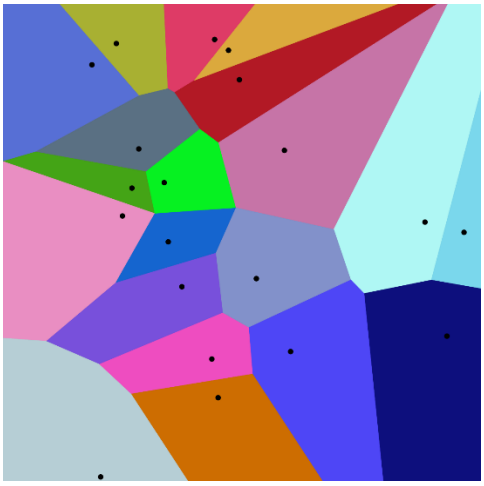


Figura 2-7. Diagrama de Voronoi.

2.1.2.5. Modelatge i simulació de sistemes complexos

Tot i que els algorismes espacials ens donen moltes opcions per crear espais utilitzant equacions, a vegades son poc útils per descriure elements naturals. Aquest problema es pot solucionar utilitzant models i simulacions.

2.1.2.5.1. Autòmat cel·lular

Un autòmat cel·lular (Chopard & Droz, 1998) és un model matemàtic que simula el comportament de les cèl·lules disposades en una quadrícula, on cada cel·la pot estar en un de diversos estats i segueix un conjunt de regles que determinen com canvia l'estat de la cèl·lula al llarg del temps en funció dels estats de la seves cèl·lules veïnes. El resultat de la simulació pot ser impredictible, però també repetitiu.

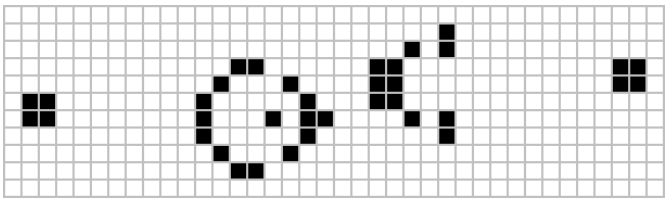


Figura 2-8. El joc de la vida de Conway, un dels autòmats cel·lulars més coneguts.

2.1.2.5.2. Camps de tensors

Els camps de tensors, en la GPC, són camps on cada punt de l'espai té assignat un tensor que s'utilitza per especificar la forma de l'espai (Chen, Esch, Wonka, Müller, & Zhang, 2008, p. 3). Els tensors són objectes matemàtics que es poden utilitzar per descriure propietats físiques, igual que els escalars i els vectors. De fet, els tensors són només una generalització d'escalars i vectors; un escalar és un tensor de rang zero i un vector és un tensor de primer rang (Rist &

McGinnigle). Utilitzant les *hiperstreamlines*, curves que son tangents al camp eigenvector en tots els seus punts, es pot visualitzar fàcilment el camp tensorial i utilitzar-lo per varies aplicacions. Per exemple la generació de la xarxa viària d'una ciutat de Chen et al. (2008).

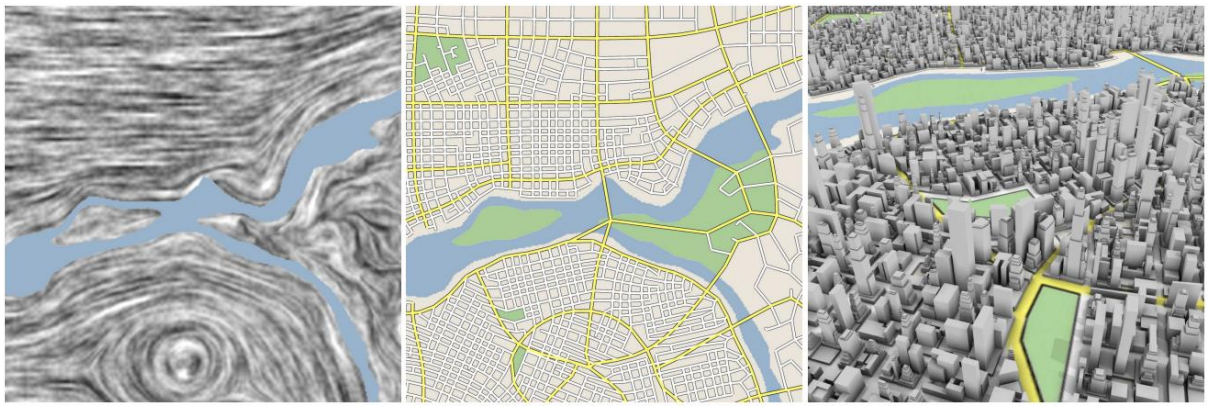


Figura 2-9. Resultats de Chen et al. (2008). A l'esquerra visualització de les hiperstreamlines, al centre la generació de la xarxa de carrers, i a la dreta el resultat final.

2.1.2.5.3. Simulació basada en agents (ABS)

El modelatge o simulació basada en agents és una manera de simular sistemes complexos utilitzant entitats individuals, anomenats agents. Aquests agents interactuen entre ells i amb el seu entorn, donant lloc a un comportament emergent. Aquest comportament sovint és difícil de predir a partir de les accions dels agents individuals, i contrasta amb comportament de les tècniques tradicionals. A més, els models basats en agents permeten la manipulació dels agents durant la simulació i la possibilitat que els agents aprenguin al llarg del temps.

2.1.2.6. Intel·ligència artificial

La intel·ligència artificial es un camp en la computació que té com a objectiu imitar la intel·ligència dels animals o dels humans. Les seves aplicacions son molt variades ja que l'objectiu final en aquest camp és precisament poder resoldre tot tipus de problemes.

2.1.2.6.1. Algorismes genètics

Els algorismes genètics (GA) (O. Kramer, 2017; S. Forrest, 1996) són un mètode per resoldre problemes d'optimització que imita el procés d'evolució biològica. Les possibles solucions (cromosomes) es representen com a cadenes de caràcters o vectors i s'avaluen mitjançant una funció que determina la qualitat de la solució (fitness).

Al ser un procés iteratiu, s'inicia creant una generació de *N* cromosomes aleatoris i per cada bucle de l'algorisme s'avaluen amb la funció de fitness per trobar les millors solucions. Per formar la nova generació hi ha tres mètodes, que normalment es combinen.

- **Selecció d'elit**: es seleccionen els cromosomes amb una puntuació de fitness més alta. Aquests cromosomes son els pares de la següent generació. En alguns casos aquests també formen part de la nova generació.
- **Encreuament:** es combinen dos pares en un punt aleatori del cromosoma per formar els cromosomes de la següent generació.
- **Mutació:** s'altera en punts aleatoris el cromosoma d'un pare per formar els cromosomes de la següent generació.
- **Mapejat de genotip-fenotip**: Dependent de la representació de la solució al cromosoma, és necessari un mapeig del cromosoma del genotip a la solució real, que seria el fenotip. Aquest mapeig genotip-fenotip evita introduir un biaix a les noves generacions.

Un cop s'ha format la nova generació es torna a avaluar amb la funció de fitness i torna a començar el bucle. Aquest bucle finalitza quan la població convergeix, és a dir, les noves generacions no presenten una millora rellevant, o quan s'arriba al màxim de repeticions determinat.

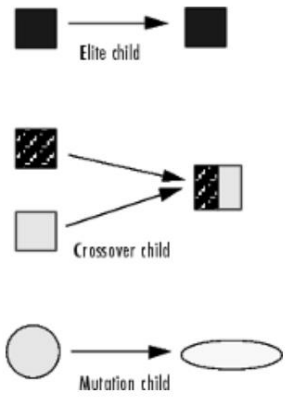


Figura 2-10. Normes bàsiques d'un algorisme genètic per crear la següent generació.

https://es.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

2.1.2.6.2. Xarxes neuronals artificials

Les xarxes neuronals artificials o ANNs (de l'anglès) (Haykin, 1998) són models computacionals que tenen la peculiaritat d'*aprendre* a partir d'un input de dades, ja sigui amb assistència o sense. Aquest aprenentatge es basa en trobar la relació entre un input i un output per tal de que l'output sigui el més proper al output esperat. Les ANNs poden fer una gran varietat de tasques: reconeixement de patrons, classificar, recordar i estructurar dades. Aquestes habilitats bàsiques, sumat a l'ús de models de *deep learning*, ha fet que en els últims anys les ANN hagin sigut la punta de llança en la investigació acadèmica de la GPC, ja que s'han vist grans avenços en models generatius capaços generar text i diàlegs creïbles (Radford, Narasimhan, Salimans, & Sutskever, 2018), imatges a partir de un input de text (Ramesh, et al., 2021), (Dhariwal, et al., 2020), vídeo (Yu, et al., 2022), models 3D (Gao , et al., 2022), animacions (Peng, Guo, Halper, Levine, & Fidler, 2022) i més.

El funcionament bàsic de les ANNs està inspirat en el funcionament del cervell. El component bàsic es la neurona i de la mateixa manera que les neurones es connecten mitjançant sinapsis dins del cervell, també ho fan en les ANNs. Aquestes connexions normalment estan organitzades per capes, per tant podem denominar la primera i la última capa com a input i output respectivament i les capes intermèdies son les que s'encarreguen de fer la computació. Cada neurona d'una capa té una sinapsi amb cada una de les neurones de la capa anterior.

Les neurones treballen amb nombres reals, per tant cada neurona calcula el seu valor executant una funció no lineal que depèn dels pesos que te assignats a cada sinapsi. Aquests pesos s'ajusten durant la fase d'entrenament i és el que permet "aprendre" a la ANN.

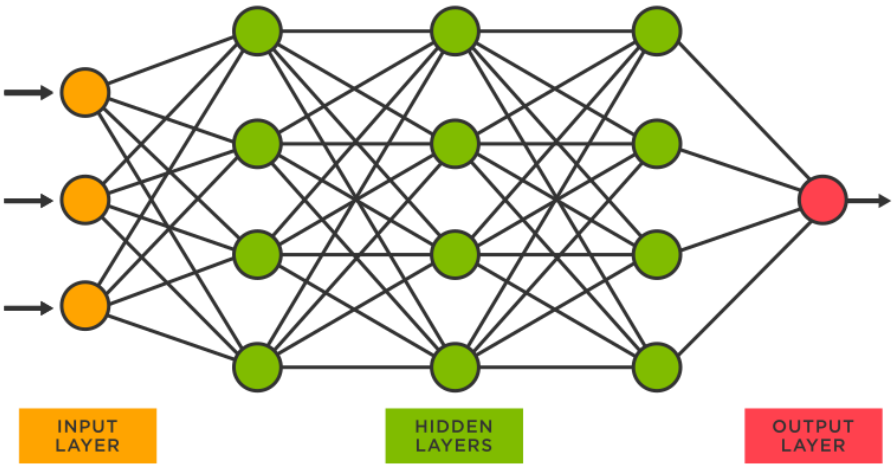


Figura 2-11. Esquema de l'estructura d'una xarxa neuronal artificial.

https://www.tibco.com/reference-center/what-is-a-neural-network

Segons Liu, Snodgrass, Khalifa, Risi, Yannakakis, i Togelius (2021) la major part de la investigació actual va destinada a la generació de nivells, principalment 2D, tot i aproximant-se des de diferents angles.

2.1.3. Representació de coneixements

Els algorismes explicats anteriorment poden ser alterats per la representació de coneixements, és a dir, les peces que l'algorisme ajuntarà per formar el contingut final. Smith (2015) proposa una categorització de la representació de coneixements en quatre blocs, la diferència entre aquests blocs es basa en qui té més pes sobre el resultat final: l'autor o l'algorisme. Els blocs son presentats de més a menys autoria humana.

2.1.3.1. Fragments experiencials

Els fragments experiencials són fragments que contenen una gran part de contingut i que per sí sols poden ser percebuts pel jugador com una sola entitat. Un exemple de fragments experiencials el trobem en la generació de nivells de *Warframe* (Digital Extremes, 2013), els nivells del joc es generen de forma procedimental enllaçant una sala darrere una altra i per tant, cada sala forma una entitat per si mateixa. El principal avantatge d'aquest model de representació de coneixements és que es reté molt control artístic i de disseny sobre el resultat final, tot i així té l'inconvenient de que per al jugador és molt evident els fragments que estan repetits. Per tant, si es vol evitar que el jugador reconegui els patrons del contingut

generat els fragments experiencials s'haurien d'evitar i utilitzar representacions de coneixement de més baix nivell o combinar-los amb altres mètodes .

2.1.3.2. Plantilles

Les plantilles son un nivell per sota dels fragments experiencials, on el dissenyador reté encara control sobre el resultat final però deixa espai al generador per poder contribuir més sobre el resultat final. Les plantilles son un bon balanç entre el control i la varietat.

2.1.3.3. Components

Al igual que les plantilles els components són patrons dissenyats per persones. En el que es diferencien és en que, a diferència dels dos blocs explicats anteriorment, els components no poden ser percebuts com a contingut per si mateixos. Per exemple, els enemics en un FPS tenen un comportament establert per un dissenyador, tot i així, aquest comportament no pot existir si no és en un nivell.

Utilitzar components redueix el risc de que el jugador reconegui els patrons en el contingut generat, això significa que l'algorisme de generació determina pràcticament la totalitat del contingut generat. Els components es poden utilitzar amb pràcticament tots els algorismes de generació procedimental explicats anteriorment.

2.1.3.4. Subcomponents

Els subcomponents son la representació utilitzant els *assets* més petits possibles, aquests no tenen significat per si mateixos ni informació de com s'ha d'utilitzar. Es poden descriure com a les peces que una persona utilitzaria per crear un nivell, com ara les peces d'un *tileset*. Molt pocs generadors utilitzen aquesta representació de coneixements, ja que sense semàntica és difícil saber com s'han d'ajuntar les peces, per exemple, el generador ha de saber quina és la relació entre un cofre i una sala per poder col·locar-los de forma correcta.

Un dels pocs exemples que hi ha d'aquesta representació de coneixements la trobem a *Galactic Arms Race* (Evolutionary Games, 2014), un joc experimental que crea diferents armes de sistemes de partícules basat en l'ús de armes en el passat. És a dir, les armes que crea el joc no són les més utilitzades en el passat sinó que genera noves armes basades en les que son més utilitzades.

2.1.3.5. Mescla i combinació

Tan les diferents representacions de coneixements com els algorismes i mètodes explicats anteriorment poden ser combinats de moltes maneres per donar lloc a sistemes més sofisticats, que permeten al dissenyador tenir més o menys control sobre el contingut generat i també permet ajustar el cost computacional dels models per tal de que puguin operar en temps d'execució.

Els paradigmes de mescla i combinació poden ser molt potents i generar moltes coses, tot i així, el seu principal problema es que les diferents capes del generador no es poden comunicar fàcilment. Per exemple, en un sistema per generar interiors que esta format per la mescla de dos sistemes, un que genera la distribució de les habitacions i un altre que les decora. Si per el que sigui una habitació no pot ser decorada per culpa de la seva distribució s'ha de tirar enrere l'algorisme i generar una nova distribució. Això passa perquè el sistema que genera la distribució de sales no coneix les restriccions del sistema que les decora.

2.2. Gramàtiques en la GPC

2.2.1. Què son les gramàtiques?

La gramàtica és la ciència del llenguatge o estudi del sistema d'una llengua determinada, tot i que aquest terme ha estat utilitzat de manera que comprenia tots els aspectes de la morfologia moderna (fonologia, morfologia, sintaxi i lexicologia), recentment, el terme gramàtica ha pres una interpretació més estricta, com una part de la lingüística que es limita a aquells aspectes que en la gramàtica clàssica s'anomenaven flexió (o morfologia) i sintaxi (Grup Enciclopèdia, 1996). Dins de la gramàtica podem diferenciar-ne diferents tipus, de les quals ens enfocarem en la gramàtica generativa i la gramàtica formal.

La gramàtica generativa (Chomsky, Syntactic Structures, 1957) és una teoria lingüística sobre la qual s'assenten tots els models generatius que utilitzen gramàtiques. Aquesta teoria descriu la gramàtica com un sistema de regles i transformacions que generen totes les combinacions de paraules que són considerades gramaticalment correctes. La gramàtica formal, també creada per Chomsky (1957) i molt vinculada a la gramàtica generativa, és el conjunt de normes capaces de generar totes les possibilitats combinatòries d'un idioma, ja sigui un llenguatge natural o un llenguatge formal.

2.2.1.1. Gramàtiques formals

Una gramàtica d'un llenguatge formal es pot definir a partir de quatre parts, una gramàtica $G = (N, T, R, I)$, on V és el conjunt que conté vocabulari complet del llenguatge, N és el conjunt que conté tot el vocabulari no terminal $N \subseteq V$, T és el conjunt que conté el vocabulari terminal $T \subseteq V$, I és el conjunt que conté el o els objectes inicials o axioma $I \subseteq N$ i R és el conjunt de normes de reescriptura (Wonka, Wimmer, Sillion, & Ribarsky, 2003), que tenen la forma $V^*NV^* \rightarrow V^*$ on * representa la clausura de Kleene, una operació unària que s'aplica sobre un conjunt de cadenes de caràcters o símbols i representa el conjunt de totes les cadenes que es poden forma a partir de qualsevol numero de cadenes del conjunt inicial, incloent repeticions, concatenacions i cadenes buides. En el cas que la cadena a la dreta de la fletxa sigui una cadena buida, normalment s'indica amb λ o ε per evitar confusions.

Per exemple, imaginem una gramàtica on $U = \{A, b, c\}$, $N = \{A\}$, $T = \{b, c\}$, com que $I \subseteq N$, $I = A$, i finalment les normes que formen la gramàtica són:

- 1) $A \rightarrow bA$
- 2) $A \rightarrow c$

En aquest cas, els símbols no terminals estan marcats amb majúscules i els símbols terminals amb minúscules. Per aplicar les normes, es substitueix el símbol de la esquerra de la fletxa per els símbols de la dreta, i es repeteix fins que tots els símbols de la cadena de caràcters siguin terminals. Per exemple:

$$A \overset{1}{\rightarrow} bA \overset{1}{\rightarrow} bbA \overset{1}{\rightarrow} bbbA \overset{1}{\rightarrow} bbbbA \overset{2}{\rightarrow} bbbbc$$

Com es pot observar, aquesta gramàtica dona lloc a un llenguatge formal on el resultat sempre serà zero o més b i finalment una c. Aquest procés on es van aplicant les normes s'anomena procés de producció o procés de derivació. Una cadena de caràcters generada utilitzant una gramàtica (G) que defineix un llenguatge formal (L) es diu que pertany a $L(G)$.

2.2.1.2. La jerarquia de Chomsky

Les normes que conformen una gramàtica poden classificar-se en una jerarquia en funció de les restriccions a les quals es sotmeten aquestes normes (Chomsky, 1959). La taula següent resumeix els quatre tipus de gramàtiques:

Gramàtica	Llenguatge	Autòmat	Normes de producció
Tipus-0	Recursivament enumerable	Màquina de Turing	$\gamma \rightarrow \alpha$ (γ no buida)
Tipus-1	Sensible al context	Autòmat linealment acotat	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Tipus-2	Lliure de context	Autòmat amb pila no determinista	$A \rightarrow \gamma$
Tipus-3	Regular	Autòmat d'estats finits	$A \rightarrow x$ o $A \rightarrow xB$

- $x =$ símbol terminal
- $A, B =$ símbol no terminal
- $\alpha, \beta =$ cadena de símbols terminals i/o no terminals o cadena buida
- $\gamma =$ cadena de símbols terminals i/o no terminals

Figura 2-12. Les restriccions de la jerarquia de Chomsky.

Cada un d'aquests tipus té una capacitat de generació diferent, per tant hi ha conjunts de cadenes que es poden generar mitjançant una gramàtica de tipus n però no amb una gramàtica de tipus $(n+1)$. Aquest fer provoca que els quatre tipus de gramàtiques estiguin niuades una dins de l'altra (Hunter, 2021).

- **Gramàtiques tipus-0.** Son gramàtiques que no tenen cap mena de restricció.
- **Gramàtiques tipus-1.** Les gramàtiques sensibles al context (CSG, de l'anglès) són gramàtiques que compleixen amb la primera restricció. Per tant la norma només pot reescriure un únic símbol, per tant una norma del tipus $aB \rightarrow Ba$ quedaria exclosa, ja que esta substituint dos símbols de la cadena. El que si que imposa la restricció és que poden ser afectades per el context, és a dir, per reescriure A es poden aplicar normes diferents depenent del que siguin.
- **Gramàtiques tipus-2.** Les gramàtiques lliures de context (CFG, de l'anglès) són gramàtiques que compleixen amb la segona restricció, és a dir, compleixen la primera però α i β (Figura 3-12) són cadenes buides. La part de la dreta de la norma pot ser una cadena de qualsevol combinació de símbols. Un exemple clàssic d'una gramàtica tipus-2 és la que genera la cadena $a^n b^n$ (on $n \geq 1$). En aquest cas el símbol inicial és S i les normes de producció són: 1) $S \rightarrow aSb$ i 2) $S \rightarrow ab$. Una derivació d'aquesta gramàtica seria:

$$S \overset{1}{\rightarrow} aSb \overset{1}{\rightarrow} aaSbb \overset{1}{\rightarrow} aaaSbbb \overset{1}{\rightarrow} aaaaSbbbb \overset{2}{\rightarrow} aaaaabbbbb$$

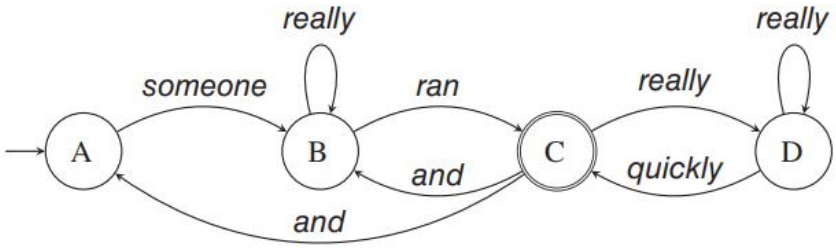
- **Gramàtiques tipus-3.** Les gramàtiques d'estats finits (FSG, de l'anglès) són gramàtiques on les normes, a més a més de ser lliures de context, la part dreta de la fletxa només pot estar formada per un símbol terminal o un símbol no terminal i un símbol terminal. Gràcies a aquesta restricció es poden representar com a màquines d'estats finits, on els estats són el elements no terminals i les transicions entre estats son els símbols terminals. Un dels estats és designat *estat inicial* (indicat amb una fletxa) i alguns dels altres estats són designats *estats acceptables* (indicat amb un doble cercle). Una FSG pot generar una cadena s si i només si existeix una sequencia de transicions que porten de l'estat inicial a un estat acceptable que genera els símbols de s en ordre (Hunter, 2021). Per exemple, la FSG de la figura 3-13, pot generar $\boldsymbol{someone\ really\ ran}$, però no pot $\boldsymbol{someone\ ran\ really}$ ja que no acaba en un

estat acceptable, ni ***someone ran quickly*** perquè no existeix un ordre de transicions per aquesta cadena.

start symbol: A

A → *someone* B
B → *really* B
B → *ran* C
B → *ran*
C → *and* A
C → *and* B
C → *really* D
D → *really* D
D → *quickly* C
D → *quickly*

(a) Rewrite-rule representation



(b) Graphical representation

Figura 2-13. Una FSG, representada com a normes de reescriptura (a) i gràficament (b).

2.2.2. Evolució de les gramàtiques i tipus d'algorismes.

2.2.2.1 Sistemes-L

Els sistemes de Lindenmayer o sistemes-L, és el primer tipus de gramàtiques generatives que es van utilitzar. Aquest mètode va ser desenvolupat per Astrid Lindenmayer (1968), un biòleg i botànic teòric de la universitat de Utrecht, que va utilitzar els sistemes-L per descriure el creixement de les plantes. Més tard els sistemes-L van adquirir varis usos dins del camp dels gràfics d'ordinador (Smith, 1984; Prusinkiewicz & Hanan, 1989; Prusinkiewicz & Lindenmayer, 1990), principalment en la generació de fractals y el modelatge realista de plantes.

La idea central dels sistemes-L és utilitzar les gramàtiques generatives (Chomsky, 1957) per generar estructures extensibles per replicar el creixement de les plantes. Totes les normes de reescriptura dels sistemes-L son almenys de tipus 1 en la jerarquia de Chomsky, ja que només operen sobre un símbol. El tipus més simple de sistemes-L és conegut com a sistema-D0L (D per determinista i 0 per ser lliure de context (CFG)) i podem entendre el seu funcionament mitjançant l'exemple que ens ofereixen Prusinkiewicz i Lindenmayer (1990).

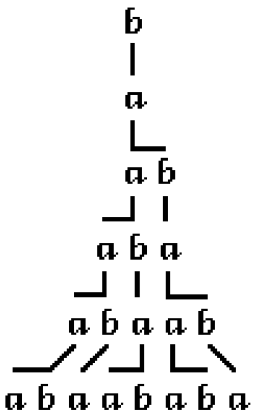


Figura 2-14. Exemple d'una derivació en un sistema-D0L.

En l'exemple (Figura 3-14) tot el vocabulari és no terminal, $V = \{a, b\}$, $I = \{b\}$ i les normes de reescriptura són $1)\ b \rightarrow a$ i $2)\ a \rightarrow ab$. Com que es tracta d'una gramàtica determinista, la producció es fa sempre d'esquerra a dreta seguint la cadena de símbols. Com que l'axioma es b, en la primera iteració només podrem aplicar la norma 2, i la cadena ens quedarà com a a. En la segona iteració aplicarem la norma 1 i la cadena resultant serà ab. En la tercera iteració ja podem aplicar les dues normes, es farà d'esquerra a dreta i quedarà la cadena aba, en la

quarta iteració quedarà $abaab$, després $abaababa$, i així fins a l'infinit. Una curiositat d'aquesta gramàtica és que la llargada de les cadenes segueix la seqüència de Fibonacci.

Utilitzant aquest sistema, però aplicant normes més complexes es poden crear una gran varietat resultats codificant en cada símbol una o més accions. Per exemple, el fractal de l'arbre binari, definit per $G = \langle N = \{0,1\}, T = \{[,]\}, I = \{0\}, R = \{(1 \rightarrow 11), (0 \rightarrow 1[0]0)\}\rangle$, les accions codificades són:

- 0: dibuixa una línia acabada en una fulla.
- 1: dibuixa una línia.
- [: *push* de la posició i l'angle actuals, rota 45º a l'esquerra.
-]: *pop* de la posició i l'angle, rota 45º a la dreta.

Push i *pop* es refereixen a una pila LIFO (últim dins, primer fora). Al iterar aquesta gramàtica, es genera un fractal que recorda a la forma d'un arbre.

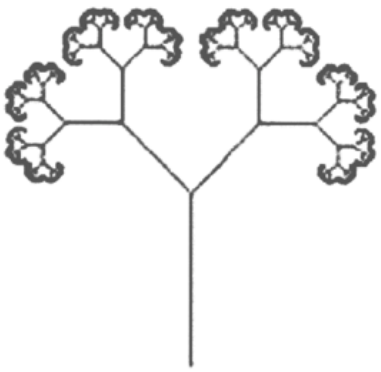


Figura 2-15. Exemple d'arbre fractal. (University of Colorado. Molecular Cellular and Developmental Biology)

Els sistemes-L portats a l'extrem, permeten generar una gran quantitat de continguts, un dels exemples més rellevants és el model original de *ArcGIS CityEngine* (2008) creat per Parish i Müller (2001).

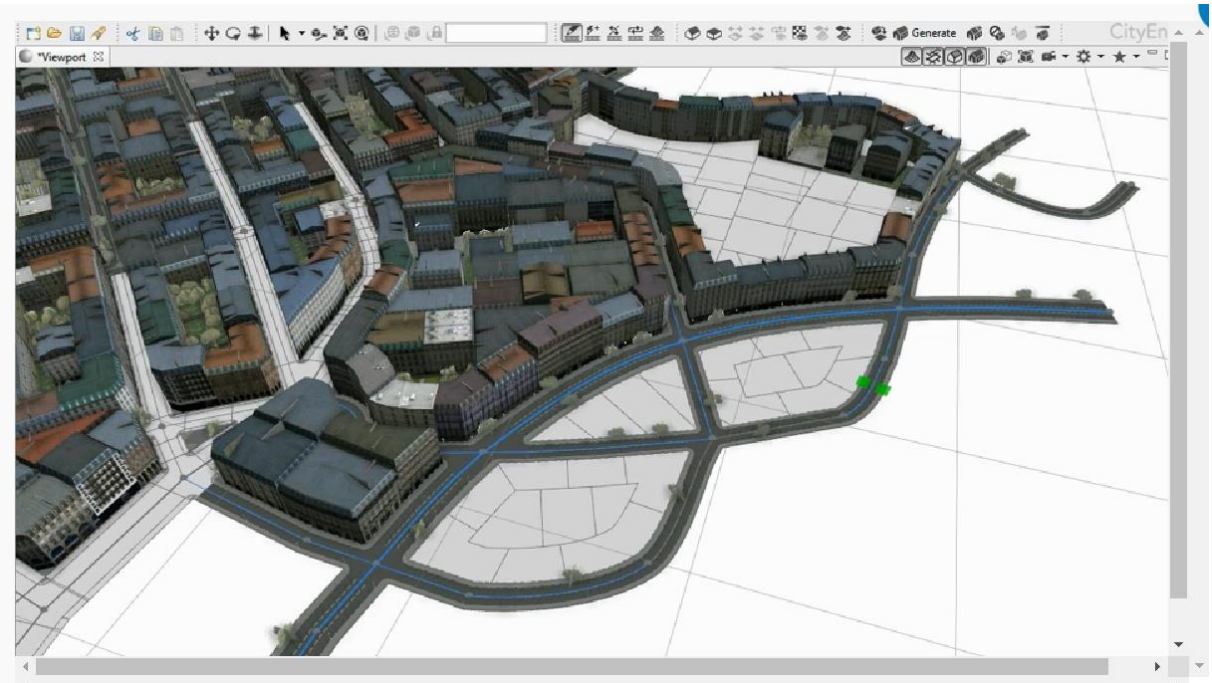


Figura 2-16. Captura de pantalla de ArcGIS CityEngine (2008).

2.2.2.2. Gramàtiques de forma

Les gramàtiques de forma (Stiny, 1975) (Stiny, Introduction to Shape and Shape Grammars, 1980) proporcionen un mitjà per la generació recursiva de formes similar a les gramàtiques d'estructura de frase, originalment utilitzades per Chomsky (1957) en lingüística. On una gramàtica d'estructura de frase es defineix sobre un alfabet de símbols i genera llenguatges de cadenes de símbols, les gramàtiques de formes es defineixen sobre alfabets de formes i generen llenguatges de formes. És a dir, en comptes d'operar sobre una cadena de símbols, opera sobre un pla bidimensional on només hi ha formes.

Una *forma* és qualsevol distribució de línies que tingui una especificació pictòrica. Una *especificació pictòrica* és qualsevol dibuix que es pugui dur a terme sobre una superfície plana d'àrea finita (p. e., un full de paper) en un període finit (p. e. amb un nombre finit de pinzellades) (Stiny, 1975). Per operar amb les formes defineix un seguit d'operacions: la unió, la translació, la rotació, l'escala i la simetria. Dues formes s'anomenen *pictòricament equivalents* si i només si les seves especificacions pictòriques són idèntiques; dues formes són *congruents* si i només si una es por fer pictòricament equivalent a l'altra utilitzant una seqüència de transformacions excloent l'escala; dues formes són *similars* si i només si una es por fer pictòricament equivalent a l'altra utilitzant una seqüència de transformacions incloent

la escala. Una forma és una *subforma* d'una segona forma si i només si l'especificació pictòrica de la primera forma és idèntica a alguna part de l'especificació pictòrica de la segona forma. Alternativament, l'especificació pictòrica de la primera forma es pot obtenir esborrant alguna part de l'especificació pictòrica de la segona forma.

Una gramàtica de formes està definida per quatre parts: $SG = \langle N, T, I, R \rangle$

- N és un conjunt finit de formes.
- T és un conjunt finit de formes de tal manera que les formes que formen part de T^+ i N^+ són distingibles. (El símbol ($^+$) és l'equivalent a la estrella de Kleene però sense incloure la forma buida).
- R és un conjunt finit de normes amb forma $u \rightarrow v$, on u i v són formes formades per la unió de formes de T^* i N^*. La forma u ha de tenir una subforma que pertanyi a N^+, la forma v pot ser una forma buida.
- I és una forma formada per la unió de formes presents a T^* o N^*. Ha de tenir almenys una subforma pertanyent a N^+.

Un exemple de les gramàtiques de forma ens el dona Stiny. Una gramàtica $SG = \langle N, T, I, R \rangle$ on:

- N conté:
- T conté:
- R conté:
- I és:

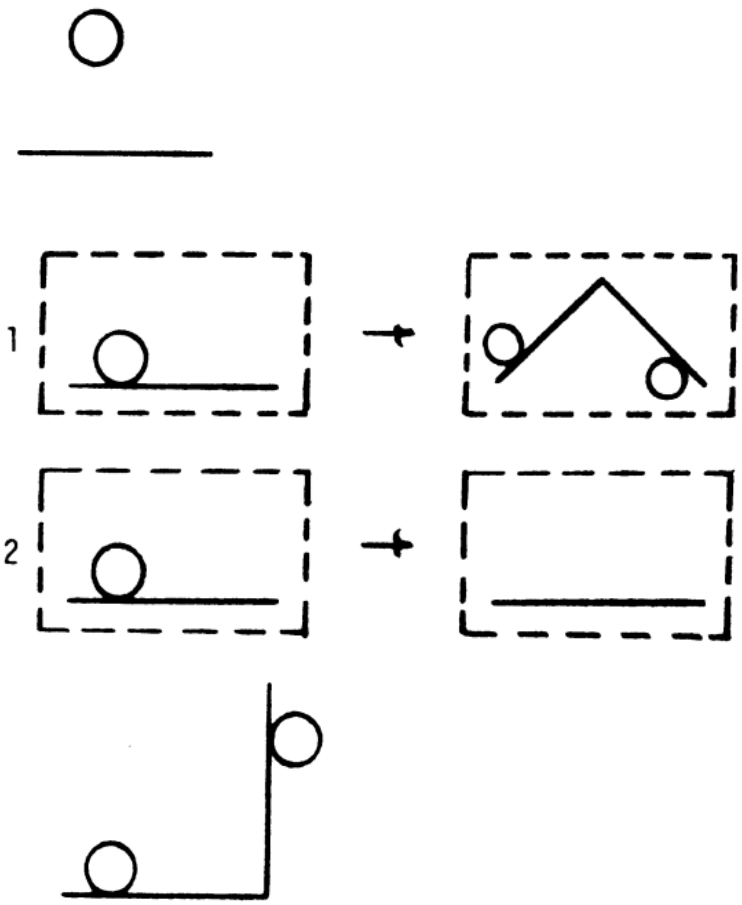


Figura 2-17. Exemple de la definició d'una gramàtica de forma.
(Stiny, 1975)

Com es pot observar a l'exemple (Figura 2-18), a la part dreta de les normes hi ha una forma (u) formada per la unió de les dues formes que conté el "vocabulari", i a la part esquerra (v) una altra combinació de formes del "vocabulari". Les normes que tenim són: la primera substitueix la forma u_1 per la forma v_1, que són dues instàncies de la forma u amb algunes transformacions aplicades, per el que ens permet seguir aplicant normes; la segona norma substitueix la forma u_2 per la forma v_2, la forma v_2 és un terminal, i per tant ja no podríem seguir aplicant més normes sobre v_2. Per fer la derivació d'aquesta gramàtica, es fa de la següent manera: primer s'identifica la o les subformes de la definida a la banda esquerra de la norma (u), en aquest cas trobem dues instàncies de u en I, per tant les dues normes que tenim es podrien aplicar a I. L'ordre d'aplicació de les normes és indiferent. Quan s'aplica una norma, es substitueix la forma de la part esquerra per la forma de la dreta. En l'exemple de derivació (Figura 2-18) podem observar com primer s'aplica varies vegades la norma 1 i finalment la norma 2 també varies vegades.

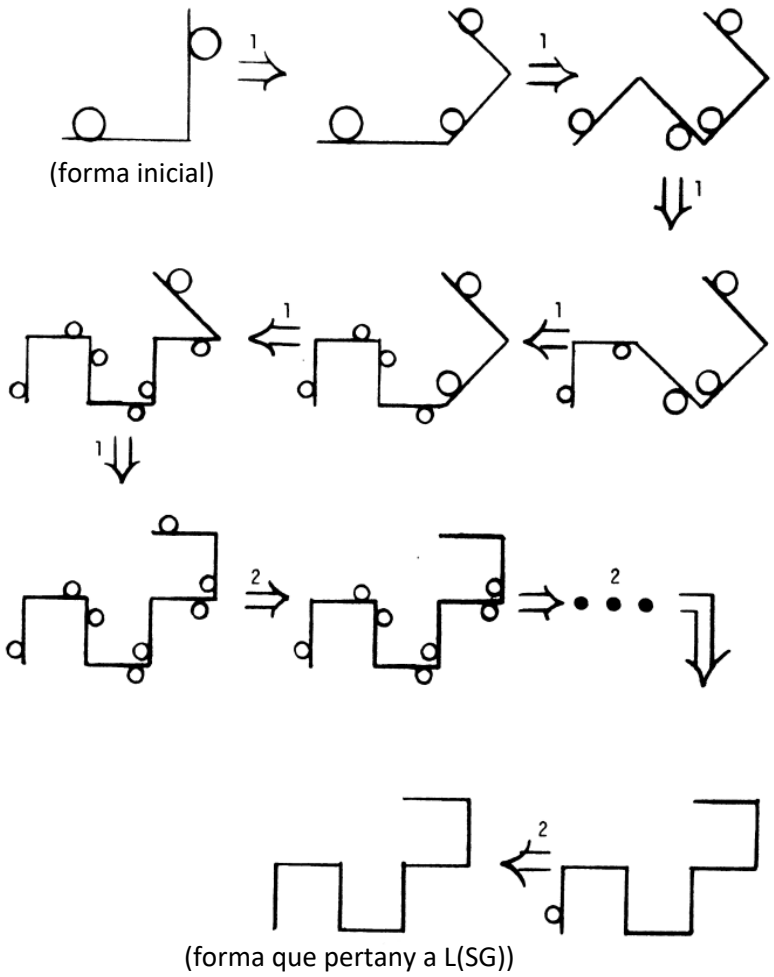


Figura 2-18. Exemple de derivació d'una gramàtica de formes.
(Stiny, 1975)

2.2.2.3. Gramàtiques de conjunts

Les gramàtiques de conjunts (Stiny, 1982) són una evolució de les gramàtiques de forma, aquest nou tipus de gramàtica empra un vocabulari que està format per un conjunt de formes i un conjunt de símbols, només elements que pertanyen al conjunt de formes poden ser terminals, i per tant, els dissenys generats en el llenguatge definit per una gramàtica d'aquest tipus no tindran cap símbol associat. Aquest petit canvi permet eliminar el concepte de subforma i permet que les gramàtiques construïdes a partir de les gramàtiques de conjunts siguin molt mes senzilles d'implementar en un entorn digital.

2.2.2.4. Gramàtiques de divisió

Les gramàtiques de divisió, introduïdes per Wonka, Wimmer, Sillion i Ribarsky (2003) són un tipus especialitzat de gramàtica de conjunts que opera sobre formes. La seva idoneïtat per al modelatge automàtic d'edificis prové del fet que s'han escollit les restriccions per tal de trobar un equilibri entre l'expressivitat de la gramàtica, és a dir, el nombre de dissenys diferents que permet, i la seva capacitat per a la selecció automàtica de regles. Els objectes manipulats per la gramàtica són certes formes atribuïdes, parametritzades i etiquetades, que anomenem *formes bàsiques*. Una forma bàsica està formada per $b = \langle s, P, V \rangle$, on s és una forma simple (objecte tridimensional convex tancat), P determina la posició, rotació i escala i V és el símbol del vocabulari que representa aquesta forma.

El nom de la gramàtica be donada per la seva operació principal: la *divisió*. Una divisió és la descomposició de un a forma bàsica en altres formes del vocabulari , per exemple una divisió substitueix un cuboide per $x \times y \times z$ cuboides, ordenats en una quadrícula definida per plans de divisió. L'altra operació rellevant és la *conversió*, on una forma bàsica és substituïda per una altra, amb la única condició de que la segona ocupi el mateix espai que la primera.

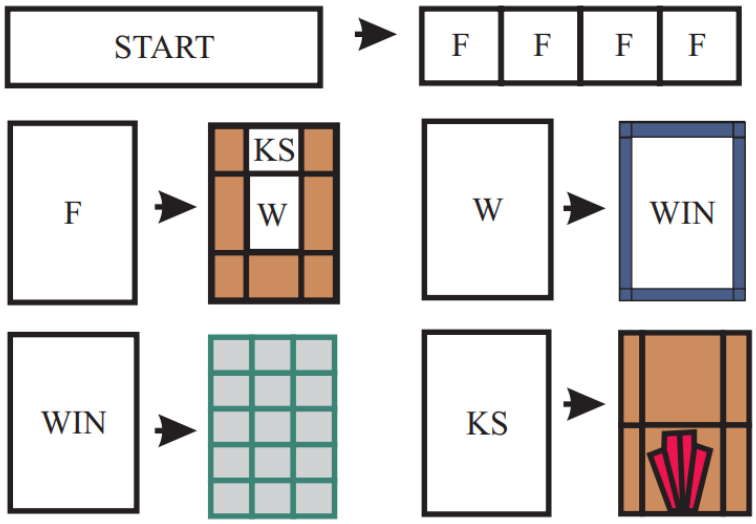


Figura 2-19. Les normes d'una gramàtica de divisió simple. Les àrees blanques (definides per símbols) representen les formes bàsiques no terminals, les àrees amb color representen les formes terminals. La figura inicial (START) es divideix en 4 elements de façana (F), cada element de façana es divideix en una finestra (W), una clau de volta (KS) i diferents elements de paret, etcètera. (Wonka, Wimmer, Sillion, & Ribarsky, 2003)

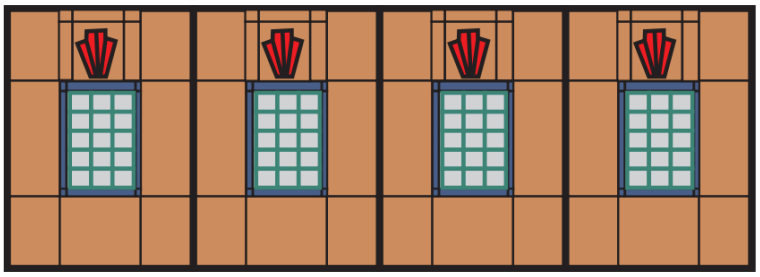


Figura 2-20. Resultat de la derivació de la gramàtica definida a la figura 2-19. (Wonka, Wimmer, Sillion, & Ribarsky, 2003)

El *framework* desenvolupat per Wonka, Wimmer, Sillion i Ribarsky (2003) no només es basa en les gramàtiques de divisió, sinó que té una estructura més complexa. Utilitza la gramàtica de divisió com a nucli, però també compta amb un sistema de concordança d'atributs que permet a l'usuari especificar varis objectius d'alt nivell i controla la aleatorietat per tenir resultats més consistents; també inclou una gramàtica de control que és una gramàtica sense context simple que s'encarrega de generar la distribució espacial dels atributs, permetent que el contingut generat s'adhereixi als principis arquitectònics i no sembli un caos aleatori.

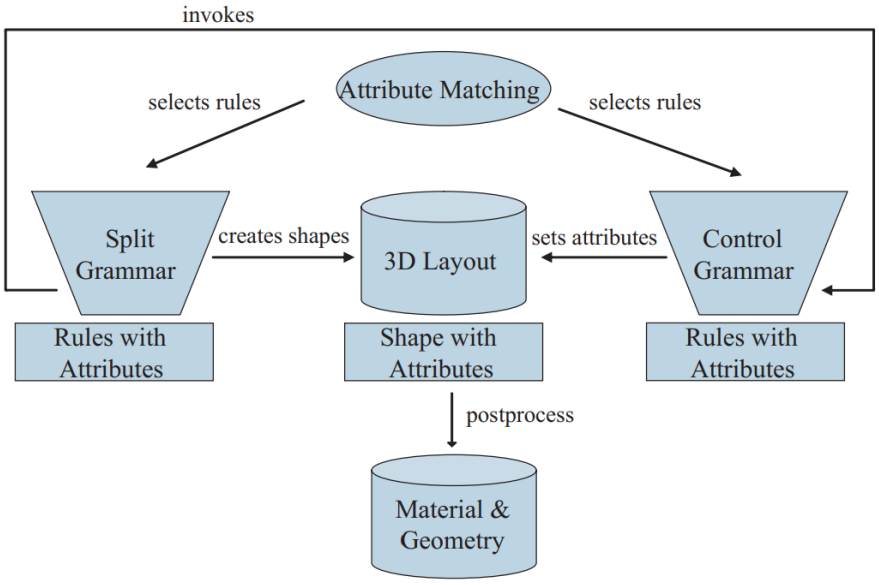


Figura 2-21. Esquema de la interaccions entre els diferents elements que conformen el *framework*. (Wonka, Wimmer, Sillion, & Ribarsky, 2003)

Com es pot veure a la figura 2-21, el procés de derivació esta controlat per les dues gramàtiques, les quals fan ús del sistema de concordança d'atributs per tal de seleccionar la norma que encaixi millor. Aquest sistema d'atributs es basa en que no només les formes, sinó

que també les normes poden tenir atributs associats. Entre les dues gramàtiques formen una derivació que esta composta per formes amb atributs assignats. Finalment, es fa ús d'un sistema d'un últim procés per convertir les formes en geometria i materials corresponents.

El procés de derivació segueix els següents passos:

1. La gramàtica de divisió busca totes les normes que coincideixin amb la forma actual.
2. El sistema de concordança d'atributs selecciona una d'aquestes normes basant-se en els atributs de la forma i de les normes candidates.
3. Es copien els atributs de la forma original a totes les formes generades per l'aplicació de la norma
4. La gramàtica de control es crida perquè distribueixi els dissenys en l'espai. La derivació de la gramàtica de control fa ús del mateix sistema de concordança d'atributs que la gramàtica de divisió.
5. La gramàtica de divisió s'invoca de forma recursiva per totes les noves formes generades en aquesta aplicació de norma.

Els sistemes de control que té l'usuari per interactuar amb el *framework* son: Primer, modificar les gramàtiques generadores (divisió i control), que tenen la influència més gran sobre el contingut generat. Segon, modificar els atributs de la forma inicial, que afecten als aspectes de més alt nivell, com ara l'estil, la edat, o l'ús de l'edifici. I per últim els atributs de les normes de les gramàtiques, que permeten un control més fi sobre alguns aspectes del disseny.

2.2.2.5. Gramàtiques de parets.

Les gramàtiques de paret (Larive & Gaildrat, 2006) són un sistema de gramàtiques enfocat a generar exteriors d'edificis. Les gramàtiques de paret es construeixen com a gramàtiques de divisió però operen sobre parets en comptes de formes. Aquest fet simplifica molt l'ús, però redueix la varietat de generacions possibles. El *framework* desenvolupat per Larive i Gaildrat (2008) juntament amb les gramàtiques de paret també fa ús de atributs com el *framework* desenvolupat per Wonka, Wimmer, Sillion i Ribarsky (2003), aquests s'assignen a les façanes i es representen com a cadenes de caràcters. Per exemple, si una façana té l'atribut "blind", no es generaran finestres, ja que la façana esta tapada per l'edifici del costat.

En les gramàtiques de paret cada element del vocabulari té una norma intrínseca assignada, però només trobem cinc elements que formen part del vocabulari. Totes les parets tenen els atributs de dimensions mínimes i màximes, i material de la paret final.

- **Panell de paret (WP).** És la forma més simple de totes i és la única forma terminal. A més a més dels atributs base, tenen una referència a un model 3D que es pot instanciar al centre de la paret, per exemple una finestra. Per les següents descripcions (W) fa referència a qualsevol tipus de paret
- **Paret emmarcada (BW).** És una paret que té quatre marges al voltant i un element central que referència a una altra paret. Els seus atributs són la mida del marge i la política d'escala, és a dir si la mida del marge és variable, fixe, no pot ser més petit de la mida, etcètera.

  $$BW \rightarrow W$$
- **Paret extruïda (EW).** La paret extruïda és una paret que fa una extrusió. El seu atribut marca la distància de l'extrusió, que pot ser positiva o negativa. La cara extruïda resultant fa referència a qualsevol tipus de paret.

  $$EW \rightarrow W$$
- **Llista de parets (WL).** Conté una llista que referència varies parets. Pot ser vertical o horitzontal.

  $$WL \rightarrow W_1 W_2 \dots W_n$$

- **Quadrícula de parets (WG).** Conté una única paret que es repetirà en una o dues direccions (horitzontal i vertical). Els atributs especifiquen la quantitat de repeticions en cada orientació.

$$WG \rightarrow W^{(1-h)(1-v)}$$

Al ser una gramàtica que permet extrusions permet crear fàcilment balcons, entrades, terrasses, etcètera.

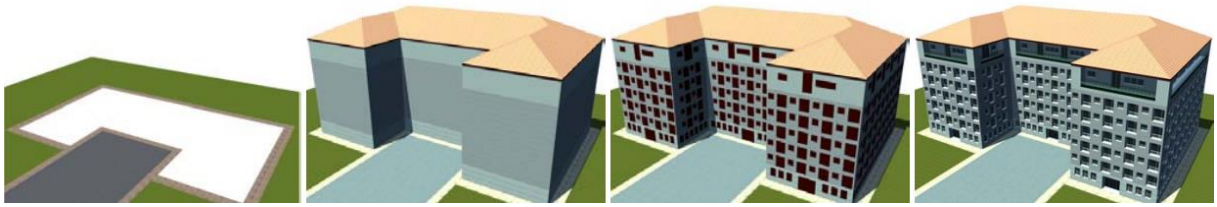


Figura 2-22. Exemple de derivació d'un edifici amb gramàtiques de paret. De dreta a esquerra: dades d'entrada (petjada de l'edifici), primer pas de la derivació, definir les àrees d'extrusió i el resultat final.

2.2.2.6. Gramàtiques de forma CGA

Les gramàtiques CGA (Müller, Wonka, Haegler, Ulmer, & Van Gool, 2006) són una evolució de les gramàtiques de divisió introduïdes per Wonka et al. (2003), però poden treballar en una, dues o tres dimensions i afegeixen altres operacions per adaptar millor el sistema al modelatge d'arquitectura. CGA significa "Computer Generated Arquitecture".

En aquest sistema una forma es defineix amb cinc parts: un símbol (cadena de caràcters) que identifica la forma, una geometria i atributs numèrics, dels quals els més importants són la posició P, tres vectors ortogonals X, Y i Z ,que descriuen un sistema de coordenades, i un vector de mida S. Aquests tres atributs defineixen una *bounding box* en l'espai anomenada àmbit. Una **configuració** és un conjunt finit de formes bàsiques.

El procés de producció es fa seguint tres passos. Des de l'axioma A, que és una configuració arbitrària de formes, s'executa de la següent forma:

1. Selecciona una forma activa B en la configuració A.
2. Selecciona una norma de producció amb B a la banda esquerra i computa el successor, un conjunt de formes B'.
3. Marca la forma B com a inactiva i afegeix a la configuració A les figures de B' i torna al pas 1.

Quan només quedin formes terminals a la configuració, el procés de producció haurà acabat. Per millorar el control sobre el resultat, les normes tenen una prioritat assignada, la següent norma a executar serà la que tingui més prioritat. Al establir les formes com a no actives al substituir-les, això permet inspeccionar la jerarquia de formes.

Figura 2-23. Exemple d'edificis generats amb gramàtiques de forma CGA.

2.2.2.6.1. Normes de producció

La notació que s'utilitza per les gramàtiques de forma CGA segueixen el següent patró:

$$id: \quad p : cond \rightarrow s : prob$$

On id és l'identificador únic de la norma, p és un símbol no terminal, $cond$ és una expressió lògica que ha de ser certa per poder aplicar la norma, s és un conjunt de formes que substituirà a p i $prob$ és la probabilitat de que es seleccioni la norma. Per exemple:

$$1: façana(h) : h > 9 \rightarrow pis\left(\frac{h}{3}\right) pis\left(\frac{h}{3}\right) pis\left(\frac{h}{3}\right)$$

En aquest cas, si el paràmetre h és més gran que 9, es substitueix per les 3 formes especificades. Existeixen quatre tipus de normes en aquesta gramàtica:

- **Normes d'àmbit.** Inclouen normes generals per modificar formes: translació T, rotació R i escala S i assignació d'una primitiva geomètrica I (p. e., cub, cilindre). Són bàsiques per definir l'axioma i les formes en general.

  $$1: A \rightarrow [T(0,0,6)\ S(8,10,8)\ I(\text{"cube"})]$$

- **Norma de divisió bàsica.** Divideix l'àmbit d'una figura en un eix i posició determinada per els paràmetres.

  $$1: façana \rightarrow Subdiv(\text{"Y"}, 3,1,3,3,3)\{pis \mid cornisa \mid pis \mid pis \mid pis\}$$

El primer paràmetre defineix l'eix, els següents les posicions on s'han de fer les divisions i per últim, s'especifiquen el tipus de forma que seran els elements dividits.

- **Norma de repetició.** Permet fer canvis en una escala més gran que les normes de divisió. Per exemple:

  $$1{:}\, pis \rightarrow Repeat(\text{"X"}, 2)\{B\}$$

  En aquest cas es repetirà B en l'eix X tantes vegades com hi hagi espai.

- **Norma de divisió de components.** Aquesta norma permet dividir les formes en els seus components, és a dir, permet crear formes de menys de tres dimensions. Amb aquesta norma es poden separar les cares, les arestes i els vèrtexs. Els àmbits d'aquestes formes estan definits amb un o més eixos de S a zero. Establint un valor a un eix amb valor zero de S permet fer una extrusió i augmentar les dimensions d'una forma.

2.2.2.6.2. Oclusió i *Snapping*

Dues característiques importants d'aquest model són la oclusió i l'*snapping*, la oclusió permet identificar si hi ha altres formes ocloent la forma actual i aplicar diferents normes segons el tipus de oclusió:

$$1{:}\ tile : Shape.occ(\text{"nonparent"}) == \text{"none"} \rightarrow window$$

$$2{:}\ tile : Shape.occ(\text{"nonparent"}) == \text{"part"} \rightarrow wall$$

$$3{:}\ tile : Shape.occ(\text{"nonparent"}) == \text{"full"} \rightarrow \varepsilon$$

El fet de poder eliminar totes les formes que no es veuran, permet augmentar el rendiment dràsticament, ja que hi ha moltes formes que ja no hauran de computar.

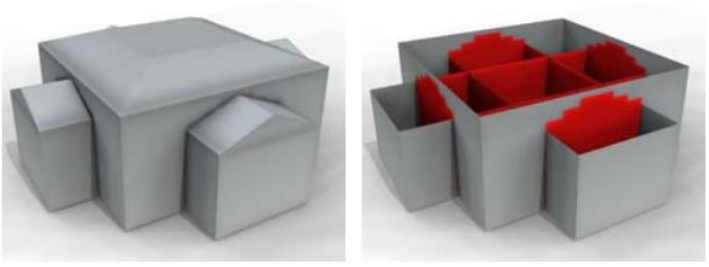


Figura 2-24. A la esquerra una configuració. A la dreta una aproximació de les cares ocloses. (Müller, Wonka, Haegler, Ulmer, & Van Gool, 2006)

Per altra banda l'*snapping* permet alterar la maquetació de la façana de l'edifici per tal de que les arestes es moguin lleugerament en el pla per tal de coincidir en diferents cares. Per calcular aquest *snaps* es calcula la intersecció planar de totes les cares de la configuració per generar

les línies, després es modifiquen les normes de divisió lleugerament per tal de que les arestes coincideixin. En la figura 2-25, es pot apreciar clarament la millora en la qualitat que ofereix aquest pas extra.

Figura 2-25. Esquerra: edifici generat sense oclusió ni *snapping*. Dreta: edifici generat amb oclusió i *snapping*.

2.2.2.7. Nodes

La evolució de les gramàtiques a partir de les gramàtiques de forma CGA s'ha basat bàsicament en simplificar l'ús d'aquestes. Les gramàtiques de forma CGA son molt potents per generar tot tipus d'edificis però només permeten interactuar indirectament amb l'edifici generat canviant la gramàtica mitjançant text. Per permetre als artistes una interacció més senzilla amb les gramàtiques, s'ha anat desenvolupant diferents mètodes de codificació de les normes de la gramàtica basats en llenguatges de programació visual basats en nodes, ja que aquests s'han fet populars entre els programes més utilitzats per artistes 3D, com ara *Maya* (Autodesk, 1998) o *Houdini* (SideFX, 1996).

Exemples de gramàtiques codificades en nodes les trobem en treballs com ara (Silva, Müller, Bidarra, & Coelho, 2013), que implementa una gramàtica molt similar a les formes CGA a través d'un sistema de nodes. També (Patow, 2012) que implementa les gramàtiques de forma CGA amb un sistema de grafs a *Houdini* (1996) per aconseguir resultats impressionants.

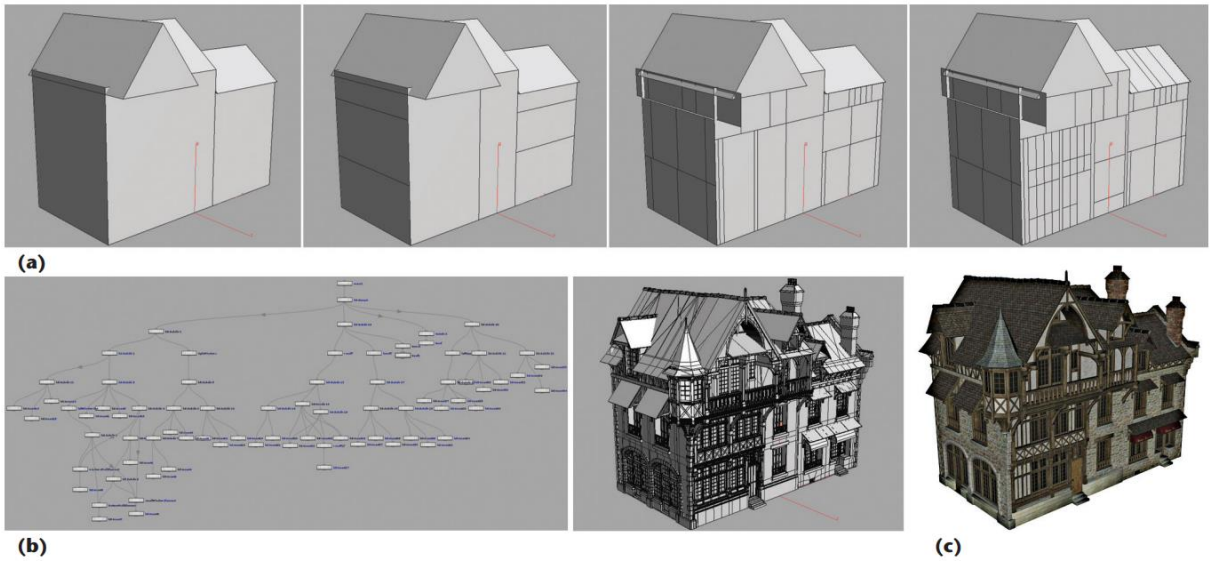


Figura 2-26. Resultats de (Patow, 2012). (a) Primers passos de derivació. (b) Part del graf i model final. (c) Model texturitzat.

3. Referents

3.1. Townscaper

Figura 3-1. Captura de pantalla del videojoc Townscaper.

Townscaper (Stålberg, 2021) és un videojoc de tipus *sandbox* desenvolupat amb Unity que ofereix al jugador la possibilitat de crear petits pobles en illes al mig del mar. El joc es basa en una quadrícula cúbica tridimensional, regular en altura però la base és "una quadrícula quadrilàtera relaxada irregular aperiòdica infinita i determinista" (Stålberg, 2019)[1]. Això significa que aquesta quadrícula està formada per quadrilàters irregulars obtinguts d'un procés de relaxació (un procés on es minimitza la llargada dels segments que uneixen els vèrtexs tenint en compte tota la malla). És irregular perquè no està formada per files i columnes, sinó que aquestes es dobleguen i s'enreden per formar formes més curioses. És aperiòdica i infinita degut a que el procés de creació permet afegir nous segments de quadrícula infinitament, però de manera que mai no es repeteixi exactament el mateix patró.

[1] Stålberg, O. [@OskSta]. (2019, 6 de juliol). *So whats your best algorithm for generating aperiodic infinite deterministic irregular relaxed quadrilateral grids?* [Tweet]. Twitter. Recollit de https://twitter.com/osksta/status/1147404919277797376

Finalment, és determinista ja que sempre es generarà de la mateixa manera utilitzant l'algorisme de generació explicat a la següent pàgina.

Per a cada casella d'aquesta quadrícula el jugador pot triar si forma part de l'edifici o no, i en el cas de que en formi part, de quin color és la façana, amb una selecció de 15 colors. El joc en si, és molt senzill i no té cap objectiu en concret.

Dos anys abans del llançament del joc, Stålberg (2019) va impartir una ponència al INDICADE EUROPE 2019, on va mostrar el prototip de *Townscaper* (2021) i va explicar el seu funcionament amb bastant detall. El funcionament és el següent:

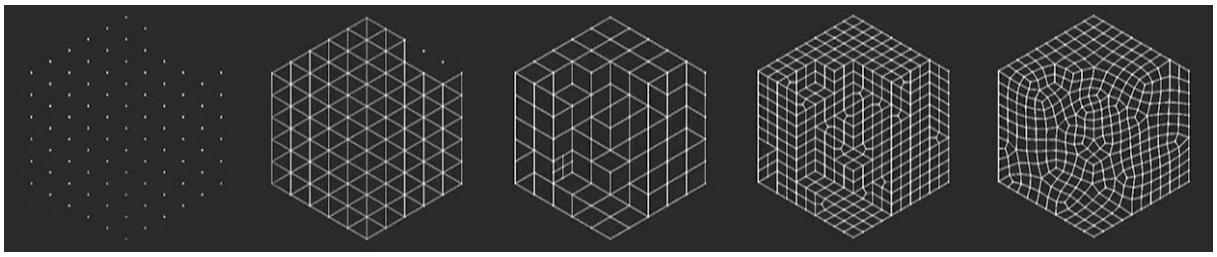


Figura 3-2. Les 5 fases, d'esquerra a dreta, de la generació de la quadricula.

El primer pas, al carregar la escena buida, genera una quadricula hexagonal gran (1) i per cada un d'aquests hexàgons, (2) genera una quadrícula triangular en seu interior. Un cop generada aquesta quadrícula, (3) s'eliminen la meitat d'arestes de la quadrícula triangular, de manera que quedi una quadrícula formada, majoritàriament, per quads. Cada un d'aquests quads (4) es subdivideix per fer més densa la quadrícula i finalment (5) es relaxa la malla. Per fer aquest procés de relaxació i tenir en compte la malla dels hexàgons veïns, es torna a executar el procés de relaxació ajuntant el hexàgons de 3 en 3, quan s'ha fet aquest procés amb tots els veïns, es determina la posició dels vèrtexs fent la mitjana de la posició de totes les solucions calculades. Amb aquest procés es crea la quadrícula base, que és el que li dona la peculiaritat característica que tenen les edificacions de *Townscaper* (2021).

Un cop generada la quadrícula el jugador ja pot interactuar amb l'escena per generar les edificacions. Les interaccions dels jugadors el que fan és definir, per cada vèrtex de la quadrícula si aquest pertany o no a la edificació. Al actualitzar l'estat de l'edificació es tornen a calcular quina peça ha d'anar a cada casella. El videojoc consta d'unes 300 o 400 peces diferents per omplir les caselles.

Per generar l'estructura es fa en tres fases, en la primera s'utilitza l'algorisme *marching cubes* per determinar quin tipus de peça encaixa amb l'estructura en aquella posició, el segon es l'algorisme *wave function collapse* per seleccionar la peça final i finalment es col·loquen les finestres i les decoracions.

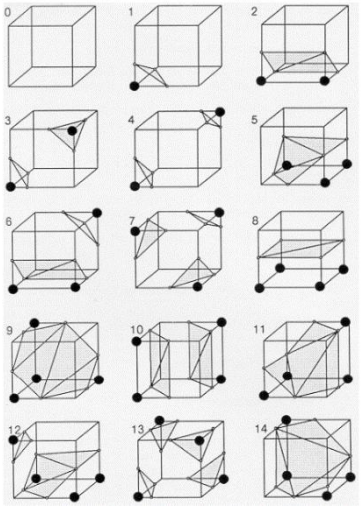


Figura 3-3. Les possibles tipologies de les cel·les en l'algorisme. (Lorensen & Cline, 1987)

L'algorisme *marching cubes* introduït per Lorensen i Cline (1987) té l'objectiu de crear una superfície de triangles de densitat constant a partir de un núvol de punts amb valor. Originalment es va desenvolupar per representar en tres dimensions les dades recollides de ressonàncies magnètiques en hospitals, però amb el temps se li han trobat aplicacions en altres àmbits. L'algorisme original parteix d'una quadricula tridimensional regular, on cada vèrtex té assignat un valor escalar. Donat un valor de tall, es discriminen quins vèrtexs queden dins i fora de la malla de triangles, i per cada casella, depenent de quins son els vèrtexs interiors i exteriors es generen els triangles necessaris per completar la malla. Existeixen $2^8 = 256$ maneres diferents de distribuir els triangles dins la casella, però si tenim en compte les permutacions possibles (simetria i rotació), es redueixen a només 15 topologies.

En el cas de *Townscaper* (2021) aquest algorisme s'utilitza per determinar quina es la topologia de la casella que ha d'omplir, però no li assigna encara cap peça, ja que per cada topologia hi ha més d'una possibilitat, per tant només redueix el nombre de peces possibles per a la casella i és el següent algorisme el que s'encarrega de triar la peça final.

L'algorisme *wave function collapse* (Gumin, 2022) es un algorisme amb l'objectiu original de crear *bitmaps* que siguin localment similars a un *bitmap* d'entrada. Localment similar significa que:

- El *bitmap* de sortida només conté els patrons existents en el *bitmap* d'entrada.
- La distribució de patrons del *bitmap* d'entrada hauria de ser similar a la distribució de patrons dels *bitmaps* de sortida. En altres paraules, la probabilitat de trobar un patró en particular a la sortida hauria de ser propera a la densitat d'aquests patrons a l'entrada.

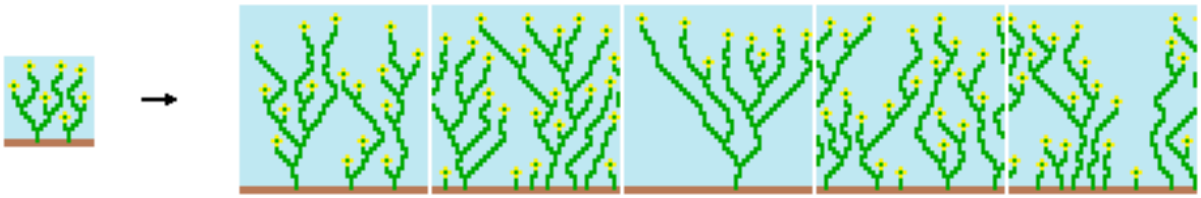


Figura 3-4. Exemple de l'algorisme original.

Des de l'algorisme original del 2016, s'ha expandit per treballar en l'espai tridimensional, i donat un *tileset* l'algorisme s'encarrega de col·locar les peces de manera que encaixin (Stålberg, 2018), és a dir s'encarrega de resoldre les restriccions que té un *tileset*. Això ho fa combinant les restriccions imposades amb algunes heurístiques per decidir la peça del *tileset* encara que hi hagi varies possibilitats d'ocupar aquell lloc. Quan es defineix una peça, s'avaluen les peces del voltant per descartar possibles solucions, cada casella que descarta una solució fa que les seves caselles veïnes també avaluïn els canvis per descartar possibles solucions.

En el cas de *Townscaper* (2021) per cada peça disponible se li assigna un valor a cada cara que defineix amb quines peces pot ajuntar-se, aquests valors defineixen les restriccions del *tileset* i li donen a l'algorisme. Utilitzar aquest algorisme permet al videojoc tenir de diferents possibles solucions per una posició determinada, per exemple, pot existir la teulada d'una casa i la teulada d'una casa amb una xemeneia, i son un conjunt d'heurístiques les que decidiran si s'utilitza una o l'altra, juntament amb l'algorisme *wave function collapse* que completarà les façanes de manera que totes les peces encaixin.

Finalment, *Townscaper* (2021) genera decoracions variades, com ara bancs, arbusts, arbres, però també les finestres i portes. Aquestes decoracions són patrons a escala més petita dissenyats per reflectir la topologia generada anteriorment. Analitzant quin es l'espai resultant al haver col·locat totes les peces a les caselles, es col·loquen les decoracions en les superfícies, ja siguin parets per a portes i finestres; terres per a bancs, plantes, fanals, etcètera; teulades per a xemeneies o finestres; o fins i tot sota els edificis, per generar suports o propulsors. Per a les portes finestres, Stålberg utilitza l'*stencil buffer* per col·locar les finestres per sobre de la malla, en lloc de tenir les finestres a les peces, ja que podrien aparèixer molts casos extrems, provocant que les finestres quedin tallades per la meitat o col·locades a les cantonades, a més d'augmentar dramàticament el numero de peces diferents que s'han de modelar.

En conclusió, *Townscaper* (2021) és un videojoc pioner en l'ús i combinació de diferents tècniques de generació procedimental per tal d'aconseguir un producte polit i interessant, que ha aconseguit triomfar sent només un editor d'escenaris.

3.2. The Architect: Paris

Figura 3-5. Captura de pantalla del videojoc *The Architect: Paris* (2021).

The Architect: Paris (Enodo Games, 2021) és un *city-builder* publicat per Enodo Games l'any 2021. Malauradament, és difícil trobar informació del funcionament del sistema de generació que s'utilitza, ja que presumptament l'estudi va fer fallida a principis del 2022, hi ha ressenyes del videojoc a Steam que ho afirmen (Nemesys9999, 2022), i el fet de que no s'hagin publicat actualitzacions del videojoc ni publicacions en xarxes socials des del desembre del 2021, fa que sigui bastant probable. En aquest videojoc se li dona la possibilitat al jugador de modificar i personalitzar els districtes centrals de la ciutat de Paris. Dins d'aquest districte, el jugador pot seleccionar i editar les illes, es poden redibuixar les diferents propietats que formen una illa i decidir què construir en cada una d'aquestes propietats. El jugador pot triar entre blocs de pisos, garatges, parcs i places.

El videojoc genera de forma procedimental el contingut de les propietats tenint en compte la seva topologia i segons els paràmetres que el jugador hagi seleccionat. Aquests paràmetres son:

- **La forma de la base:** El jugador la especifica mitjançant la eina de divisió de propietats, la façana de l'edifici sempre arriba al límit de la propietat.

- **El nombre de pisos:** El jugador ajusta els pisos amb un *slider* la quantitat de pisos que té l'edifici. El màxim son 50 i el mínim depèn del disseny seleccionat, normalment entre 1 i 4.
- **La aparença de la façana:** El jugador pot seleccionar entre una gran varietat d'aparences de façana, des de la clàssica "Haussmann" (en honor a Georges Eugène Haussman, l'autor de les extenses renovacions de la ciutat al segle XIX), fins a versions ultra-futuristes. El videojoc les classifica en dues maneres, la primera es l'estil, que es divideix en clàssic (dissenys des de medievals fins al 1920), modern (del 1920 al 1980), contemporani (del 1980 en endavant) i futurista (dissenys basats en ciència ficció). L'altre classificació el videojoc l'anomena "índex PPP" i es divideix en neutral, persones (dissenys amb més color), planeta (dissenys més sostenibles que incorporen fusta i plantes a la façana) i benefici (dissenys més estèrils, amb poc color i angles rectes).

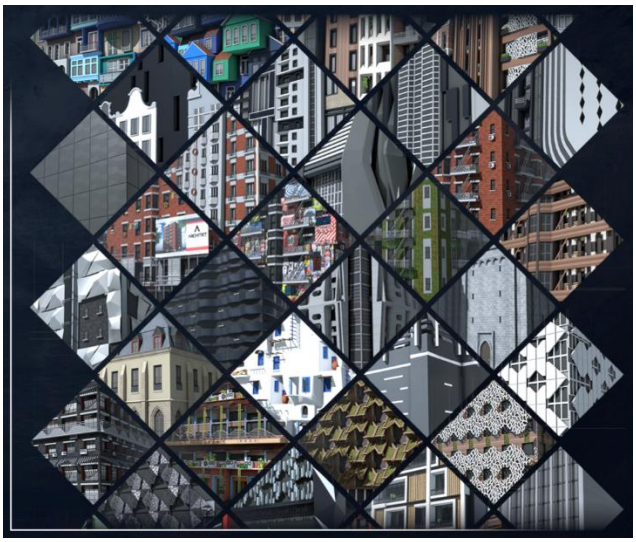


Figura 3-6. Exemples d'estils de façanes.

- **Teulada:** el jugador pot triar entre 6 estils de teulada i personalitzar també el material del qual estan fetes, en el cas de que hi hagi un terrat, seleccionar que hi ha al terrat entre algunes opcions. També es poden activar o desactivar les xemeneies.
- **Extrusions de la façana:** El videojoc permet activar les extrusions de la façana, es pot triar tan horitzontalment com verticalment, però només es pot controlar el patró periòdic que mantenen, seleccionant quants pisos o finestres sobresurten i quants no.

Figura 3-7. Exemple d'extrusions de façana.

Respecte al funcionament intern del sistema de generació no hi ha cap publicació o ponència on s'expliqui, però es poden inferir varis aspectes analitzant el contingut generat. Els edificis que genera l'algorisme, indiferentment de la aparença de la façana, es poden dividir en 4 parts fonamentals que es comporten de forma diferenciada: la coberta (ja sigui teulada, terrat o mansarda), la planta baixa, la planta superior i els pisos intermedis.

El primer que es diferencia és la teulada, té la mateixa forma que la base i és horitzontal per tapar la part superior de les façanes. Quan la teulada és inclinada, les aigües d'aquesta es divideixen per proximitat, és a dir, en qualsevol punt de la teulada sempre hi haurà pendent cap a la façana més propera. Això s'aplica quan es una teulada amb una sola inclinació i quan hi ha una teulada de mansarda amb una altra teulada a sobre. Per generar els terrats segueix el mateix mètode de generació que els parcs i les places, un sistema que col·loca les decoracions sobre la superfície plana, molt probablement seguint alguna gramàtica formal.

Cada façana està formada per una quadrícula, la mida de la qual està determinada per el nombre de pisos (vertical) i per els metros de base (horitzontal). La mida horitzontal de les cel·les varia lleugerament per ajustar-se la mida de la base, la mida vertical depèn de la aparença de la façana que hagi seleccionat el jugador. Aquesta quadrícula es pobla amb diferents tipus de portes, finestres o balcons seguint unes gramàtiques amb les quals el

jugador pot interaccionar. La façana es pot dividir en 3 segments diferenciats que es comporten de forma diferent.

La planta baixa es pobla amb elements que estan pensats per fer contacte amb el terra, sempre son diferents dels que trobem en els pisos superiors. El jugador no pot interaccionar amb aquest nivell. Per cada façana amb una mida horitzontal de més de dues cel·les es genera una porta d'entrada, en les altres caselles es genera la façana de la planta baixa o la façana de la planta baixa i l'entresol, depenent de l'aparença seleccionada.

Els pisos intermedis són els que el jugador pot modificar, el joc els pobla seguint una de 7 gramàtiques disponibles, les quals son: aleatori (s'assigna a les cel·les un fragment de façana aleatori), tauler d'escacs (s'alternen les finestres i els balcons com un tauler d'escacs), vertical aleatòria (el fragment de la façana coincideix en tots els pisos verticalment, però es elegit aleatòriament), vertical ordenada (el fragment de la façana coincideix en tots els pisos verticalment, però es selecciona seguint un ordre periòdic), horitzontal aleatori (tots els fragments d'un pis son iguals, cada pis és aleatori) i horitzontal ordenada (tots els fragments d'un pis són iguals, però segueixen un ordre periòdic).

Amb els pisos intermedis el jugador també pot fer que es generin extrusions a la façana (la façana sobresurt aproximadament un metre cap enfora). El jugador pot triar si vol que es faci horitzontalment o verticalment i establir un patró periòdic: X cel·les sobresurt, Y cel·les no ho fa.

El nivell superior es pobla amb fragments de façana sobre les quals el jugador té un control reduït, ja que son poques les aparences disponibles i normalment són la mateixa finestra o balcó repetit. Les ocasions en les que varia és degut a la interacció del jugador amb les gramàtiques que defineixen els pisos intermedis, ja que hi ha alguna cel·la que requereix una terminació específica a l'últim pis.

Cal destacar que les cel·les de les cantonades de les façanes, son diferents per tenir en compte possibles decoracions exclusives de les cantonades.

En general, és un sistema de generació que dona molta llibertat al jugador per establir la forma de la base però és bastant restrictiu a l'hora de generar les façanes. Per generar una

combinació concreta de finestres només es pot fer canviant la aparença de la façana i tornant-la a generar repetidament fins que aparegui.

The Architect: Paris (2021) és per tant un bon referent a nivell de les capacitats d'un generador, que encara que no estigui confirmat, és molt probable que utilitzi algun tipus de gramàtica.

3.3. Eines de Unity

Existeixen per al motor Unity un seguit d'eines i *frameworks* que implementen la GPC utilitzant diferents algorismes. En aquest punt s'analitzaran alguns dels més prominents. Cal tenir en compte que l'anàlisi s'ha fet a partir de la documentació únicament, ja que totes les eines analitzades son de pagament.

3.3.1. Procedural Generation Grid

Figura 3-8. Exemples d'espais generats utilitzant *Procedural Generation Grid*.

Procedural Generation Grid (FImpossible Creations, 2022) és una eina que conté solucions per un ampli nombre de generació procedimental per tal d'accelerar el procés de crear nivells per un videojoc. Aquesta eina destaca per la seva capacitat per generar i decorar espais interiors, tot i que també pot generar-ne d'exteriors.

El generador està basat en un sistema de quadrícula, que s'encarrega de col·locar tots els elements al seu lloc corresponent, seguint un seguit de normes que s'anomenen "field setup". Aquest element està format per "mods" que son els que s'encarreguen de generar els elements finals. Els "mods" actuen com a capes del generador i determinen l'ordre d'execució, per tant quan s'apliqui un "mod" ja existiran tots els elements creats per les capes anteriors. D'aquesta manera es pot controlar fàcilment la prioritat de la col·locació dels elements.

Un exemple de les capes seria, una primera per generar el terra, la segona per generar les parets, la tercera per generar esquerdes a les parets i una final per generar les llums de la sala.

Per cada "mod" l'usuari elegeix els elements a col·locar, estableix amb què s'alineen amb un sistema d'etiquetes propi, estableix normes de on es pot generar l'element i també aleatorietat en la col·locació.

Aquest referent s'ha elegit per la seva interfície, amb la qual l'usuari pot editar fàcilment les normes que dictaminen com es generaran els nivells. També podem establir una relació de similitud entre els "mods" i les normes de producció d'una gramàtica.

3.3.2. ProGen

Figura 3-9. Exemple d'edificis generats utilitzant *Progen*.

ProGen (Dilmer Games, 2020) és una eina que permet generar edificis en base a una quadricula. Aquests edificis es generen d'acord a un objecte que determina les dimensions del edifici (base i altura) i també l'estil que determina la estètica final de l'edifici.

En aquest model de generació, al ser completament basat en una quadrícula, per cada pis, hi ha un nombre indefinit de sales que son les caselles que ocupa el pis, cada sala mira quines

celes té al voltant i genera les parets d'acord amb els seus veïns. Aquestes parets poden tenir variacions, com ara portes, finestres, balcons i altres modificacions. La eina en si es senzilla i fàcil d'utilitzar, però no ofereix varietat en la generació.

Aquest referent s'ha elegit degut a que utilitza gramàtiques per definir les característiques generals de l'edifici.

3.3.3. Building Crafter

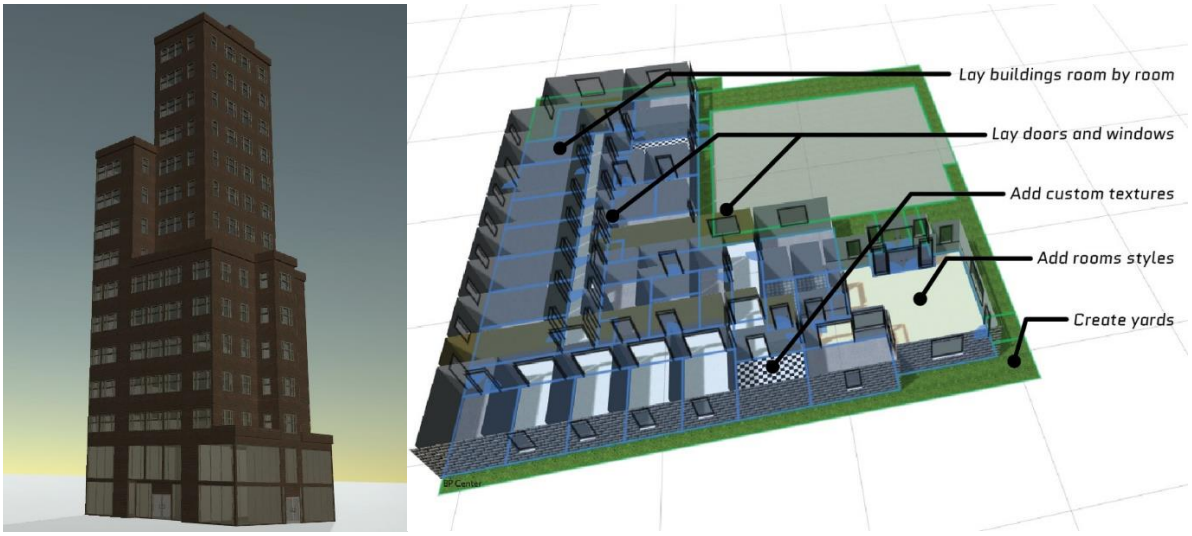


Figura 3-10. Exemple d'edificis generats utilitzant *Building Crafter*.

Building Crafter (8bit Goose Games, 2020) és una eina que cedeix a l'usuari un gran control sobre el resultat final, ja que en lloc de controlar els paràmetres de generació, l'usuari construeix sales com si fos *Els Sims* (Electronic Arts, 2000). Es marquen a la quadrícula per on van les parets, on van les finestres, les portes, les escales i jardins.

Finalment l'usuari pot personalitzar la estètica de l'edifici canviant materials, models de portes o finestres, establint quines son les façanes que donen al carrer i quines no, afegint teulades o modificant la corona en un terrat, que es genera a partir d'un dibuix 2D extruït. El que cal remarcar és que no té gaire detall, ja que està pensat com una eina de creació d'interiors per *shooters* en primera persona de mòbil.

Utilitzant aquesta eina, també pots exportar els edificis generats com un fitxer .fbx per poder utilitzar-los en altres programes, com ara Maya o Blender.

4. Objectius

Els objectius principals d'aquest treball son:

- Definir una gramàtica formal de tipus-0 a la jerarquia de Chomsky que pugui ser fàcilment implementada en C#.
- Crear un *framework* per el motor Unity que permeti crear edificacions utilitzant la gramàtica definida i utilitzar tot tipus de *assets* fàcilment intercanviables per definir l'aspecte visual de les edificacions.
- Crear la documentació necessària perquè tothom pugui utilitzar el *framework*.
- Crear una interfície de Unity per interactuar fàcilment amb el *framework*.

Els objectius secundaris d'aquest treball son:

- Polir el *framework* per publicar-lo com a eina a la AssetStore de Unity.
- Capacitar el *framework* creat per exportar fàcilment els edificis generats com a .fbx.

5. Disseny metodològic i cronograma

5.1. Metodologia

Per dur a terme el desenvolupament d'aquest treball, ja que el període de desenvolupament és curt, la metodologia ha de permetre minimitzar el risc. És per aquest motiu que es seguirà la metodologia *Agile*.

La metodologia *Agile* és un conjunt de mètodes de producció que es basen en el *Manifest per al desenvolupament àgil de programari* (Beck, et al., 2001), aquests mètodes prioritzen individus i interaccions per sobre de processos i eines, programari que funciona per sobre de documentació exhaustiva, col·laboració amb el client per sobre de negociació de contractes, resposta al canvi per sobre de cenyir-se a una planificació. Segons (Abrahamsson, Salo, Ronkainen, & Warsta, 2002), una metodologia de desenvolupament *Agile* és incremental (versions petites de programari, amb cicles ràpids), cooperativa (client i desenvolupadors treballant constantment amb una estreta comunicació), senzilla (el mètode en si és fàcil d'aprendre i modificar, ben documentat) i adaptativa (capaç de fer canvis d'últim moment).

Dins d'aquest conjunt de metodologies *Agile* en aquest projecte s'aplicarà la metodologia *Scrum,* una metodologia que es basa en la idea de la impredictibilitat, i que per tant està pensat per poder adaptar-se a tot tipus de circumstàncies (Abrahamsson, Salo, Ronkainen, & Warsta, 2002). Aquest mètode es divideix en tres grans fases:

- La **fase pre-joc**, es divideix en dues sub-fases. En la fase de planificació es defineix el sistema a desenvolupar i es crea una llista anomenada "Product Backlog" que conté tots els requeriments del projecte, esta ordenada per prioritat i s'assigna un cost de producció a cada un dels requeriments. En la fase d'arquitectura, es planifica el disseny d'alt nivell del sistema i la seva arquitectura en base als elements de la llista que s'ha fet anteriorment.
- La **fase de desenvolupament**, és la part que més s'adhereix a la metodologia *agile* ja que és la fase on més sorpreses pot haver. Aquesta fase es divideix en *sprints*, cicles iteratius on la funcionalitat és desenvolupada o millorada en petits increments. Cada *sprint* inclou les cinc fases tradicionals del desenvolupament de software: requeriments, anàlisi, disseny, evolució i lliurament. Els *sprints* acostumen a durar de

una setmana a un mes. Al final de cada *sprint* es lliura un producte funcional i es planifiquen els elements de la llista de *backlog* per l'*sprint* següent.

- La **fase post-joc**, marca el final del desenvolupament, on ja no s'afegeixen nous elements en la llista de *backlog*. En aquesta fase les tasques a dur a terme son la integració de software, *testing* i documentació.

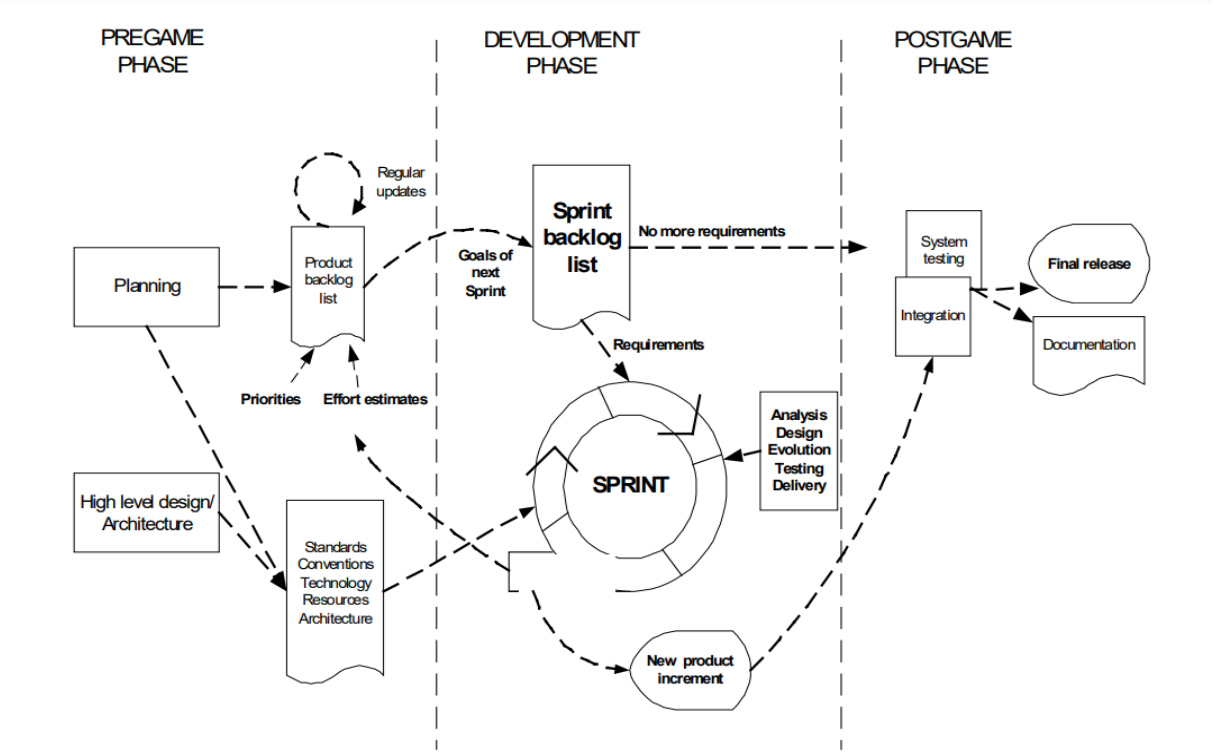


Figura 5-1. La metodologia Scrum.

5.2. Cronograma

En aquest apartat es presenta el cronograma de la elaboració del projecte.

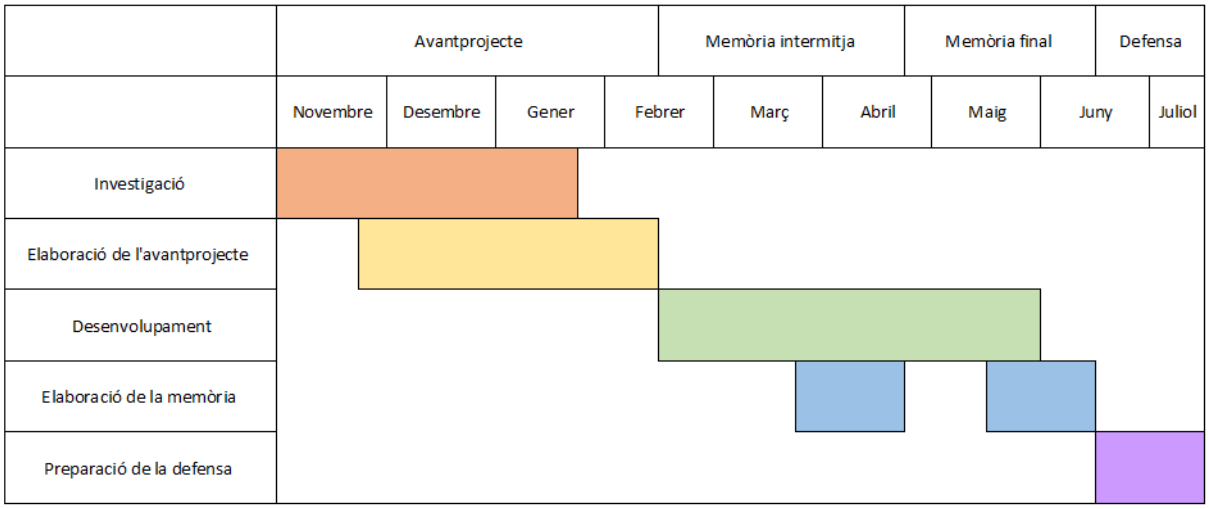


Figura 5-2. Cronograma del projecte.

6. Desenvolupament

En aquest apartat es detalla el desenvolupament i funcionament teòric del sistema que s'ha desenvolupat.

6.1. Característiques de l'eina

L'eina desenvolupada treballa dins l'entorn de Unity (Unity Technologies, 2005) i per tant tota la seva funcionalitat s'aprofitarà de la base que representa aquest motor. Les característiques que té l'eina creada són:

- L'algorisme de derivació: s'encarrega de seleccionar i aplicar les normes de la gramàtica.
- Gramàtica amb normes editables des de la interfície de l'editor.
- Creació de normes i formes de manera senzilla.
- Representació visual del resultat de la derivació, abans de tenir el resultat final de la post-producció.
- Interfície personalitzada per editar de forma senzilla les gramàtiques. Permet crear, eliminar, duplicar i editar normes. També permet crear de forma senzilla les formes que s'utilitzaran en la gramàtica.
- Petita eina que facilita a l'usuari la preparació dels models 3D per al procés de post-producció.

6.2. Gramàtica generativa

La gramàtica generativa que s'ha utilitzat en aquest projecte és una combinació de les característiques de les gramàtiques de divisió (Wonka, Wimmer, Sillion, & Ribarsky, 2003) i les gramàtiques de forma CGA (Müller, Wonka, Haegler, Ulmer, & Van Gool, 2006), bevent de la simplicitat de les gramàtiques de divisió a l'hora de definir les normes, però dividint aquestes en diferents tipus, de la mateixa manera que ho fa la gramàtica de forma CGA.

6.2.1. Formes

El vocabulari d'aquesta gramàtica està constituït per formes. Les formes poden ser de tipus tridimensional o bidimensional, però ambdues comparteixen el mateix tipus de definició basada en les formes de les gramàtiques de forma CGA.

Una forma està definida per un símbol (cadena de caràcters), geometria (model 3D), un booleà per indicar si la forma és terminal o no i finalment atributs numèrics. D'aquests, els mes importants són la posició (vector tridimensional), la orientació (quaternions) i escala (vector tridimensional). Però també n'existeixen d'altres: la escala ideal, el numero de dimensions de la forma i un booleà que indica si la forma està activa o no.

De la mateixa manera que les gramàtiques de forma CGA, l'espai cuboide que és definit per la posició, rotació i escala es denomina àmbit. El punt des d'on s'estableix la posició i serveix com a eix de rotació es denomina arrel.

Seguint la literatura establerta en les gramàtiques de forma, un conjunt de formes s'anomena configuració.

6.2.2. Procés de producció o derivació

El procés de producció treballa sobre una llista finita de formes (LF). Aquest procés pot començar amb qualsevol configuració de formes actives no terminals arbitrària que anomenarem axioma (A), és a dir, al principi del procés $A = LF$. El procés de producció es fa de forma recursiva, per a cada forma de A, de la següent manera:

1. Donada una forma activa (B) de LF es comprova si és terminal, en cas de que no ho sigui es prossegueix.

2. De les possibles normes que es poden aplicar a B se'n selecciona una utilitzant una heurística.
3. Es computa el successor de B, un conjunt de formes anomenat B'.
4. Es marca la forma B com a no activa i s'afegeixen totes les formes de B' com a filles de B en una jerarquia i s'afegeixen a LF.
5. Per cada forma de B' es torna al pas 1.

Quan a LF no queden més formes actives no terminals el procés de producció acaba. En la configuració resultant en LF existeix una jerarquia explorable de quines formes han donat lloc a cada una de les formes terminals, ja que cada forma és filla de la forma predecessora. Aquesta característica permet entendre més fàcilment quin ha sigut el procés de derivació.

6.2.2. Normes de producció

Les normes de producció estan definides per:

- El tipus de norma o operador: defineix l'operador que estarà actuant sobre la forma. Estableixen quin és el comportament de la norma i quins paràmetres requereix.
- El predecessor (cadena de caràcters): defineix quina és la forma no terminal que pot substituir.
- La llista de successors (formes), poden ser una o més formes de qualsevol tipus. Aquests successors poden ser seleccionats aleatòriament d'una taula de que indica la probabilitat de cada successor.
- Paràmetres de la norma: depenen del tipus de norma i defineixen les característiques de les formes que es produeixin.

Per poder descriure les normes de producció d'aquesta gramàtica clarament s'utilitzarà les següent notació:

$$Predecessor \rightarrow Operador(Paràmetres)\ \{Successor \mid Successor \mid Successor\}$$

6.2.2.2. Normes de tipus Àmbit (scope)

El resultat és una única forma a la que se li pot haver aplicat una translació (T), rotació (R) o escala (S). L'escala inclou canvis de dimensions i es representarà com a zero en l'eix on no existeixi la dimensió. En el següent exemple es mou i s'escala una forma.

$$A \rightarrow Scope\big(T(-2,0,1), S(2,2,2)\big)\ \{B\}$$

La norma de tipus d'àmbit es pot utilitzar també per substituir una forma per una altra sense canviar el seu àmbit i així aprofitar la elecció aleatòria d'un successor per donar varietat a la derivació.

En els tres paràmetres de la norma podem definir si els valors que donem es sumen als valors existents o reescriuen els valors existents. En la translació i rotació podem definir si aquestes operacions es realitzen en l'espai global o l'espai local de la forma. I finalment en la operació d'escala es pot definir si els paràmetres son additius o multiplicatius.

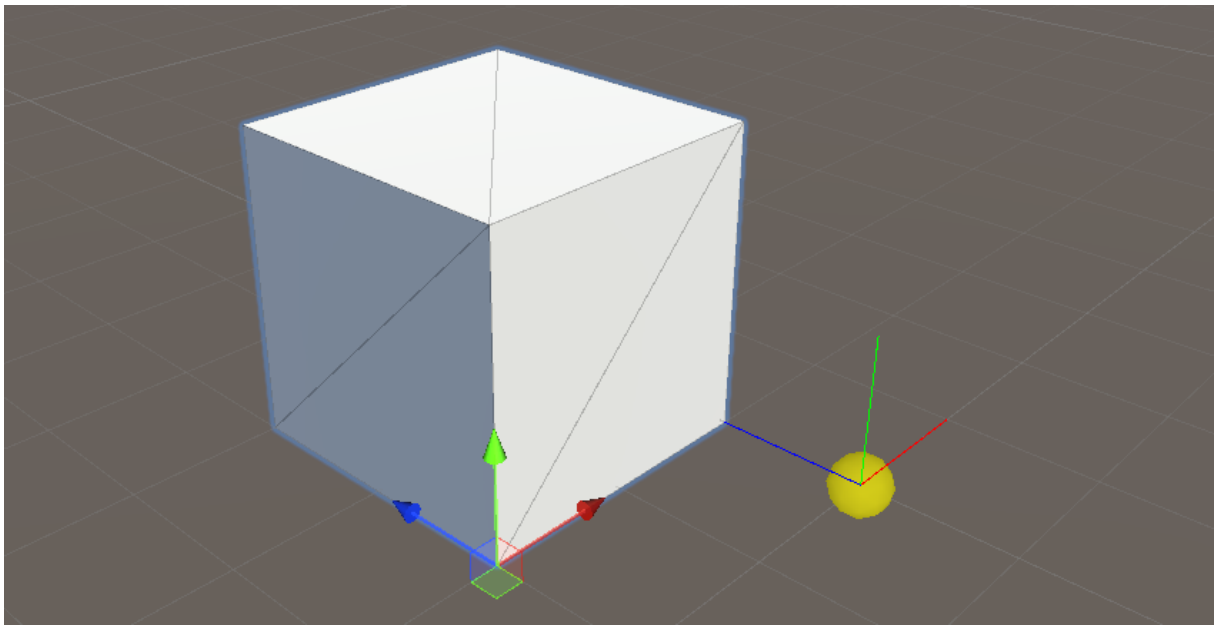


Figura 6-1. Exemple de norma de tipus àmbit. Exemplifica l'àmbit de la forma definit per la norma anterior.

6.2.2.3. Normes de tipus Divisió (split)

El resultat és un conjunt de formes la unió de les quals ocupa la totalitat de l'àmbit de la forma predecessora. Els paràmetres que defineixen com s'ha d'aplicar la norma són:

- L'eix que serà perpendicular als plans que defineixen els talls. Els eixos son els eixos locals de la forma predecessora.
- Llista de valors que defineix la escala de les formes resultants en l'eix seleccionat. Ha de ser un menys que la llista de successors perquè coincideixin els talls.

- Booleà que defineix la direcció en la que es fan els talls. En cas de ser cert es marca com a “FromRoot” i farà que els talls es facin de l’arrel cap a la resta de la forma. En cas de ser fals es marca com a “ToRoot” i els talls es faran cap a l’arrel.

En el següent exemple, la forma A es divideix en el pla definit per l’eix X de la forma i es faran els talls des de l’arrel cap a la resta de la forma.

$$A \rightarrow Split("X", \{3, 3, 2, 3\}, "FromRoot")\ \{B|C|D|E|F\}$$

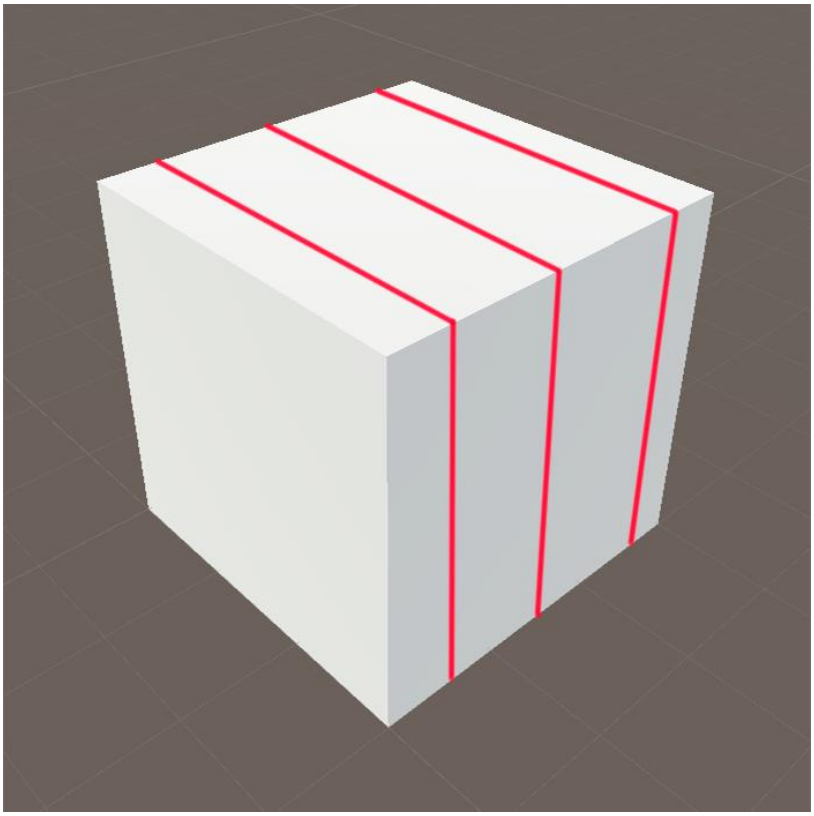


Figura 6-2. Exemple de la aplicació de la norma de tipus divisió. La forma predecessora (cubica), és dividida en quatre formes. Els talls estan marcats en vermell.

6.2.2.4. Normes de tipus Repetició (repeat)

La norma de repetició té un funcionament molt similar a la divisió, però si tornem al anàleg dels operador matemàtics si la norma de divisió és la suma, la norma de repetició és la multiplicació. És a dir tots els successors seran del mateix tipus i la distància entre talls s’ajustarà a l’escala preferida de la forma successora. Per tant l’únic paràmetre que es necessita serà l’eix que defineix el pla de divisió. Per exemple:

$$A \rightarrow Repeat("Y")\ \{B\}$$

En l'exemple, es dividirà la forma A en tantes formes B com hi càpiguen en l'espai que ocupa, és a dir, si la forma A ocupa 8 unitats en l'eix Y i l'escala preferida de la figura B en l'eix Y és de 2 unitats, llavors la figura B es repetirà quatre vegades. En cas de que la divisió no sigui perfecta s'ajustarà perquè la mida de B s'apropi el màxim possible a la seva mida preferida.

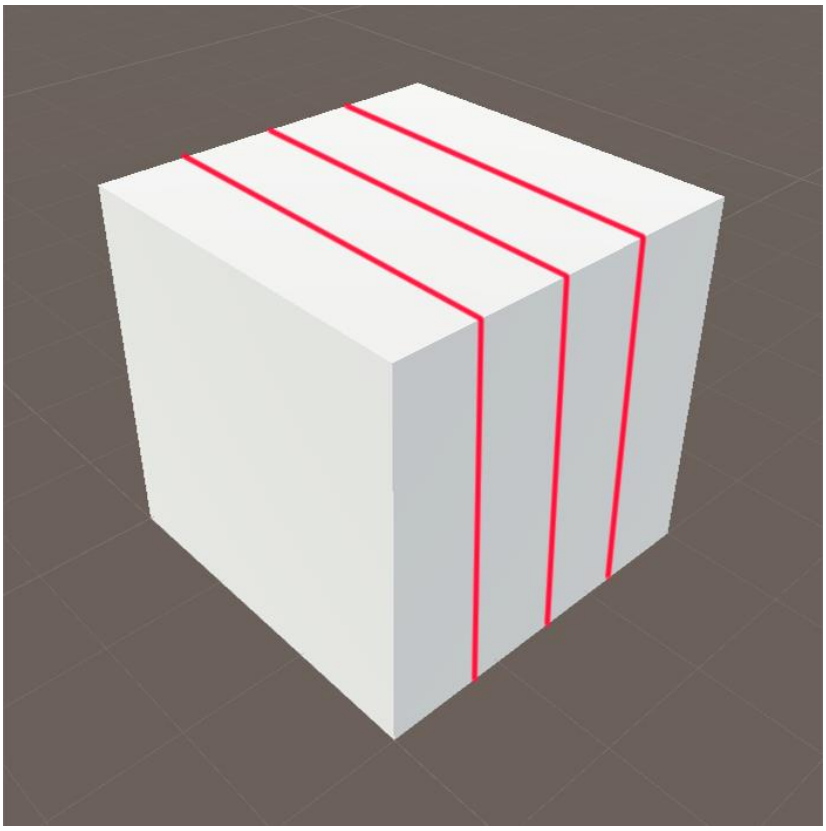


Figura 6-3. Exemple de l'aplicació d'una norma de repetició. Les línies vermelles indiquen les divisions. Les quatre formes resultants comparteixen escala.

6.2.2.5. Normes de tipus Quadrícula (grid)

Les normes de quadrícula s'inspiren en la funció del mateix nom de les gramàtiques de paret (Larive & Gaildrat, 2006) i funcionen de la mateixa manera que ho fan les de repetició però dividint la forma en dos plans diferents a la vegada. L'eix que es proporciona és l'eix paral·lel als dos plans de divisió. Els dos plans de divisió son perpendiculars. Per exemple:

$$A \rightarrow Grid("Z")\ \{B\}$$

En l'exemple, en les cares de la forma que son perpendiculars a l'eix Z quedarà una quadricula on l'escala les cel·les s'aproximarà a l'escala preferida en X i Y de la forma B.

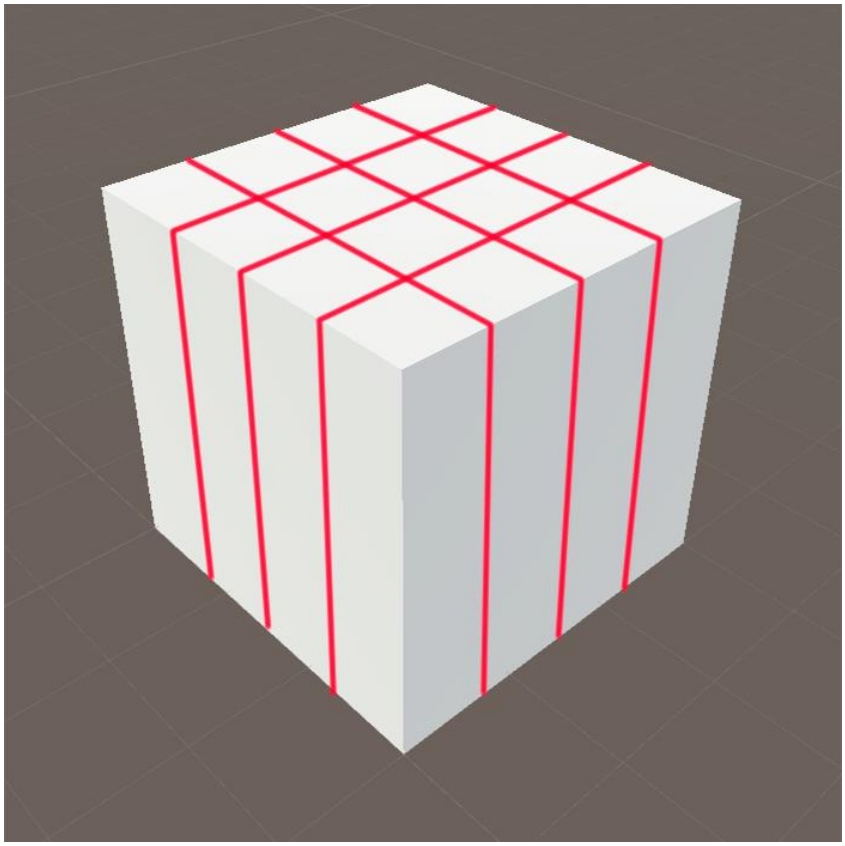


Figura 6-4. Exemple de l'aplicació d'una norma de quadrícula. Les línies vermelles indiquen les divisions. Totes les formes resultants comparteixen escala.

6.2.2.6. Normes de tipus Marge (margin)

Les normes de marge també s'inspiren en les gramàtiques de paret, però en aquest cas en la paret emmarcada. El resultat és la divisió de la forma predecessora en fins a nou formes que estableixen un marge format per cantonades, arestes i la peça central. Els paràmetres d'aquest operador són:

- Eix: l'eix és paral·lel als plans de divisió
- Marge (float): defineix la mida del marge
- Tipus de marge: defineix si la mida del marge és absoluta o relativa.
- Orientació de les cantonades: defineix la orientació de les cantonades, per exemple totes mirant cap enfora.
- Incloure la peça central (booleà): si es desactiva no es generarà la peça central.

$$A \rightarrow Margin("X", 3.5, "Absolute", "FacingOut", true)\{CORNER|EDGE|CENTER\}$$

En aquest exemple $CORNER$ seria la forma de les cantonades, $EDGE$ la forma de les arestes i $CENTER$ la forma central, si la forma central està desactivada no cal afegir-la a la llista de successors.

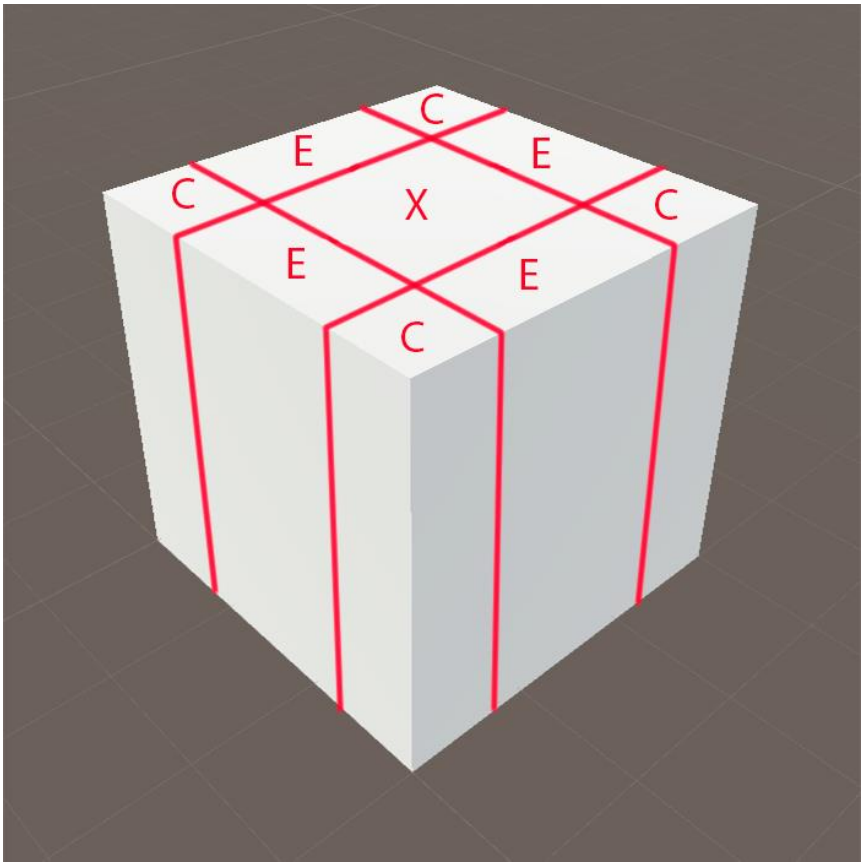


Figura 6-5. Exemple de l'aplicació d'una norma de marge. Les línies vermelles indiquen les divisions. C son les cantonades, E son les arestes i X és la peça central

6.2.2.7. Normes de tipus Component (component)

La norma de component divideix una forma en els seus components, divideix un cub en les seves cares. Cada una d'aquestes cares serà una forma bidimensional. Els paràmetres d'aquesta norma són:

- L'eix: defineix l'orientació per el pas següent:
- Mode de divisió: defineix quines son les cares que donaran lloc a una forma nova, les opcions són la superior, la inferior, les laterals i qualsevol combinació d'aquestes.

$$A \rightarrow Component(Y, Sides)\ \{SIDE|TOP|BOTTOM\}$$

Les formes resultants s'especifiquen depenent de l'ordre de la llista de successors.

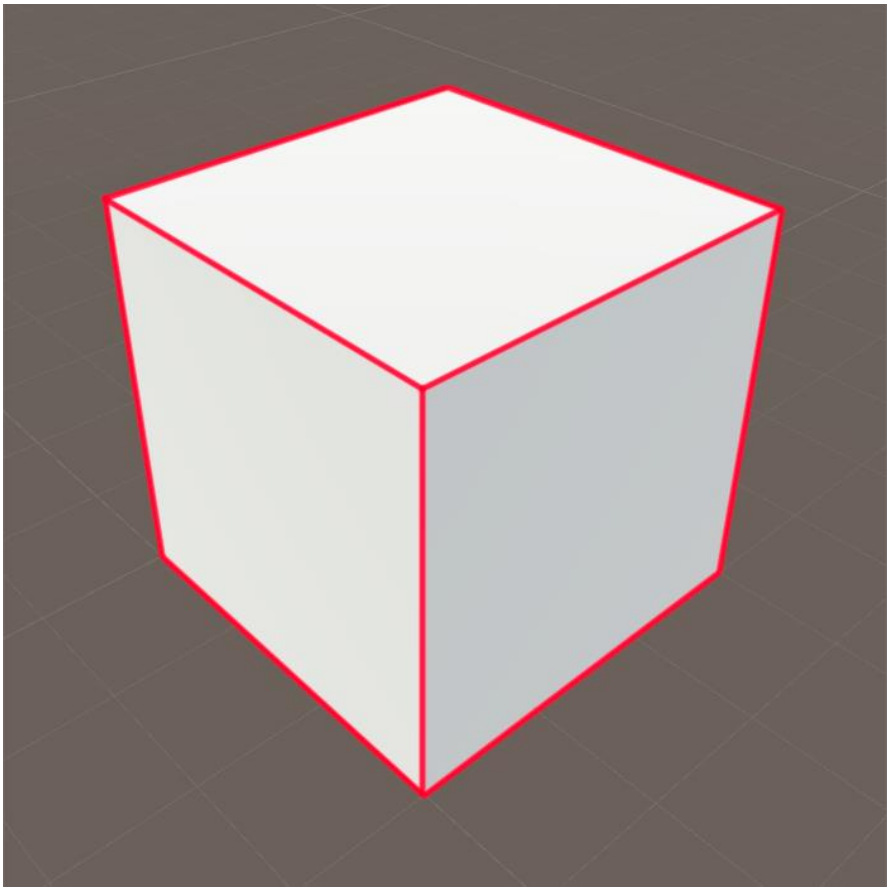


Figura 6-6. Exemple de la aplicació d'una norma de component. Les línies vermelles indiquen les divisions. Les formes resultants son les cares de la geometria.

6.2.3. Post-producció

Un cop ha acabat el procés de producció, queden les formes terminals a la llista LF. Aquesta llista és utilitzada per el mòdul de post-producció per convertir totes les formes terminals en la geometria final. Per fer-ho s'utilitza un diccionari per relacionar el símbol de la forma amb la geometria que la substituirà. La geometria final hereta de la forma que ha substituït la posició, rotació i escala, i es situa com a filla en el la jerarquia de derivació, marcant com a inactiva la forma terminal.

6.3. Implementació de la gramàtica a Unity

Per implementar aquesta gramàtica a Unity s'ha fet utilitzant scripts alguns dels quals hereten tant de *MonoBehaviour* i altres que ho fan de *ScriptableObject*. La classe *DerivationBuilder* s'encarrega d'executar el procés de derivació i postproducció, el vocabulari de formes està definit per *prefabs* que son instanciats per *DerivationBuilder*. Finalment les gramàtiques i normes de producció són objectes escriptables. A continuació s'explica amb més detall el funcionament part per part, així com una visió general de l'estructura en la Figura 6-7.

L'usuari interactua amb el *framework* utilitzant les interfícies creades, que criden les funcions pertinents i modifiquen i creen els fitxers on es guarda tota la informació de les normes, formes, gramàtiques, etcètera.

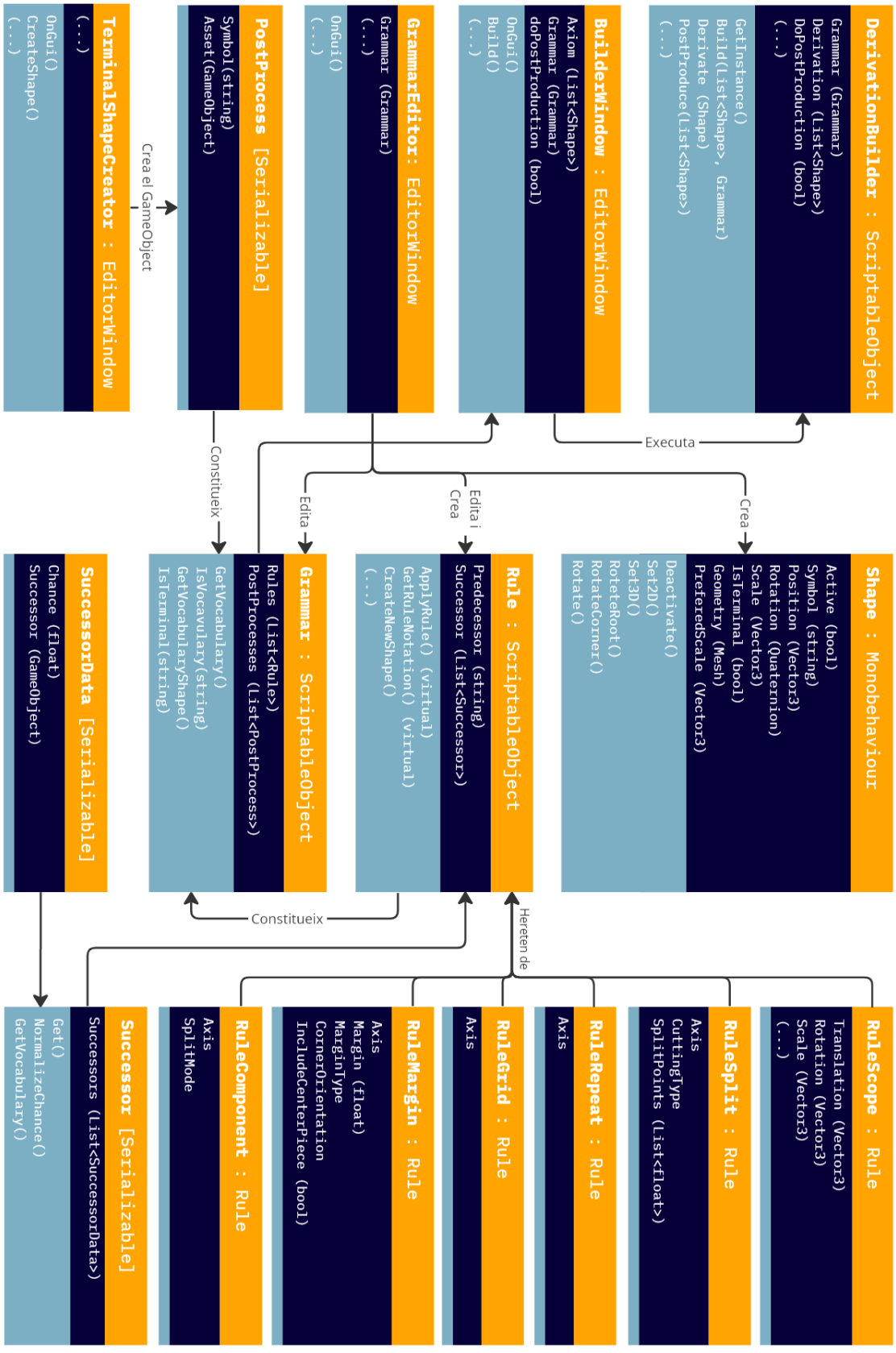


Figura 6-7 Diagrama de classes del framework. En groc els noms de les classes, amb blau fosc les propietats publiques i amb blau cel les funcions.

6.3.1. Formes

Les formes estan definides com a *prefabs* els quals consisteixen en dos *gameObjects*. El primer defineix l'arrel de la forma i les seves propietats ja que conté el component de *Shape*. El segon objecte és fill del primer i conté la geometria que representa la forma visualment en la escena. Aquesta estructura permet que el *gameObject* pare pugui guardar la informació de la posició, rotació i escala en el seu component de *Transform* i per tant aquests atributs s'ajusten de forma automàtica en l'escena. Per activar i desactivar les formes es desactiva l'objecte fill, impedint així la visualització de les formes desactivades a l'editor.

Per tal de facilitar la creació de formes noves s'ha creat un script que permet crear *prefabs* nous utilitzant la propietat de Unity *MenuItem*. Aquesta característica permet crear una forma tridimensional, una bidimensional i una buida a la carpeta del projecte que estigui oberta en aquell moment.

6.3.2. Normes de producció

Les normes de producció hereten totes d'una classe anomenada *Rule*. Aquesta defineix les propietats de predecessor i successor, la funció *CreateNewShape()* que permet a les normes instanciar formes noves, i finalment les funcions virtuals *ApplyRule()* i *GetRuleNotation()*. Les funcions virtuals s'han d'implementar a les classes filles i s'encarreguen respectivament de establir el comportament de la norma i de generar la notació especificada anteriorment per la norma en concret tal com està definida. Cada una de les classes filles de *Rule* implementa un tipus de norma i té els paràmetres d'entrada necessaris per executar-la correctament.

La classe *Rule* hereta de *ScriptableObject* i això li atorga les propietats especials de ser un objecte escriptable i per tant es molt senzill fer-ne instàncies utilitzant la propietat de Unity *CreateAssetMenu*.

6.3.3. Normes de post-producció

Per gestionar la post-producció s'ha creat un nou tipus de normes que son les normes de post-producció. Tot i així, la implementació d'aquestes dista molt de les normes de producció, ja que les normes de postproducció no implementen la seva funcionalitat, ja que d'això

s'encarrega l'algorisme de post-producció. És per això que només guarden la informació del símbol del predecessor, el qual ha de ser una forma terminal i el *prefab* de la geometria final.

6.3.4. Gramàtica

La gramàtica, així com les normes de producció, també hereta de `ScriptableObject`, però és molt més senzilla que les normes, ja que aquesta està formada únicament per una llista de normes de producció i una altra llista de normes de post-producció. El vocabulari de formes que s'utilitza en una gramàtica bé donat per les referències que contenen les normes a els successors o les formes que formen part de l'axioma.

La classe `Grammar` implementa les funcions: `GetVocabulary()`, que retorna una llista amb tots els símbols que pertanyen a la gramàtica; `IsVocabulary(string)` que determina si un símbol pertany o no a la gramàtica; `GetVocabularyShape()` que retorna una llista amb totes les formes (referències als objectes escriptables) del vocabulari i finalment `IsTerminal(string)`, que comprova si una forma és terminal o no.

6.3.5. Derivació

La execució de l'algorisme principal es fa de la mà de l'script `DerivationBuilder`, que s'encarrega d'executar els algorismes principals. Aquesta classe funciona com un *singleton* i les seves funcions es criden des de la interfície "Builder". L'algorisme de derivació s'ha implementat utilitzant una funció recursiva que es crida per cada forma de la configuració inicial i la seva propietat recursiva fa que s'executi també per cada una de les formes descendents de les formes inicials fins a trobar-se amb les formes terminals, creant així un arbre de derivació, on cada norma és filla del seu predecessor.

La execució de la funció recursiva funciona de la següent manera (Figura 6-8): la funció rep per paràmetre la forma predecessora, el primer que fa la funció és mirar si aquesta forma és terminal, en cas de que ho sigui retorna i acaba la recursivitat. Si la forma no és terminal es fa una llista amb totes les normes que tenen com a predecessora la forma que s'ha passat per paràmetre utilitzant la funció `FindRules(Shape)`, si no hi ha cap norma possible la funció retorna i salta un error per indicar a l'usuari on es troba el problema.

Per seleccionar una d'aquestes normes per aplicar es fa utilitzant una funció delegada, on l'usuari pot subscriure la funció que implementi la heurística que més convingui. S'han implementat les heurístiques: "retorna la primera norma" i "retorna una norma aleatòria"

Un cop seleccionada la norma es crida la seva funció ApplyRule(Shape) que retorna una llista amb totes les formes noves que s'han creat al aplicar la norma. Després la forma predecessora es desactiva i s'elimina de la llista de derivació activa. Finalment, per cada una de les formes noves que s'han creat al aplicar la norma la funció es crida a si mateixa però passant per paràmetre les noves formes.

```
1     private void Derivate(Shape shape)
2     {
3           if (shape.IsTerminal)
4                 return;
5           List<Rule> validRules = FindRules(shape);
6           if (validRules.Count <= 0)
7           {
8                 Debug.LogError("No rule for non-terminal shape: " +
      shape.Symbol + ".");
9                 return;
10          }
11          Rule selectedRule;
12          selectedRule = selectRuleDelegate(validRules);
13          List<Shape> newShapes = selectedRule.ApplyRule(shape);
14          derivation.Remove(shape);
15          shape.Deactivate();
16          if (newShapes.Count > 0)
17          {
18                foreach (Shape newShape in newShapes)
19                {
20                      derivation.Add(newShape);
21                      Derivate(newShape);
22                }
23          }
24    }
```

Figura 6-8. Implementació de l'algorisme de derivació.

En el moment en que totes les normes que queden a la llista de formes actives son terminals l'execució de la funció recursiva haurà acabat i llavors es passa al següent pas que és el procés de post-producció.

6.3.6. Post-producció

Per dur a terme el procés de post-producció es recorre la llista de formes actives i per cada una d'aquestes s'instancia la malla final. El model 3D ha d'estar preparat en un *prefab* específicament perquè al heretar la posició, rotació i escala de la forma terminal quedi en la

posició correcta. La relació entre les formes terminals i els models 3D es fa seguint les normes de post-producció.

6.3.5. Ordre de fitxers

Per tal de mantenir un ordre en els fitxers o *assets* que es creen al editar una gramàtica, la eina desenvolupada s'encarrega d'ordenar-los automàticament en la carpeta "GBBG_Assets".

Dins d'aquesta carpeta es creen altres carpetes amb el nom de cada gramàtica, i dins de la carpeta de cada gramàtica hi ha carpetes per els fitxers de les normes, les formes i els models finals. Les normes de post-producció son serialitzats i es graven en el fitxer de la gramàtica.

6.3.6. Interfícies

Per interactuar fàcilment amb el *framework* s'han creat tres interfícies que faciliten la edició de gramàtiques i normes, la preparació de les formes i la execució de l'algorisme.

6.3.6.1. Editor de gramàtiques

L'editor de gramàtiques és la interfície més important de les tres, ja que a través d'aquesta es gestionen les normes d'una gramàtica i s'editen els paràmetres de cada una d'aquestes normes. La interfície de l'editor de gramàtiques es pot dividir en quatre elements principals: el selector de gramàtiques, la barra d'eines, el selector de normes i l'inspector de normes.

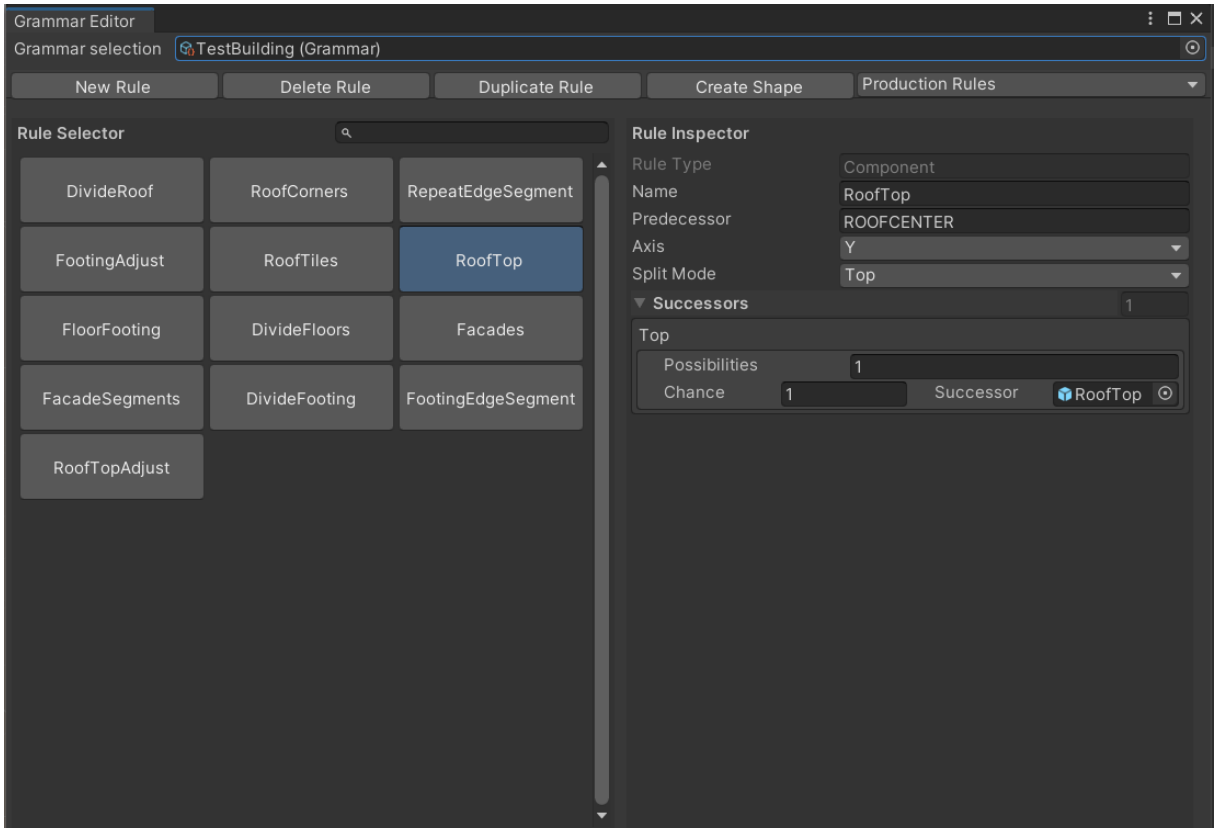


Figura 6-9. Interfície de l'editor de gramàtiques.

El selector de gramàtiques és la part més simple de la interfície, ja que només és un camp on es selecciona l'objecte escriptable de la gramàtica que es vol editar.

La barra d'eines, situada just sota el camp de selecció de gramàtica, està composta per cinc botons, cadascun amb una funció diferent:

- **Crear norma:** Crea una forma nova del tipus que l'usuari seleccioni en un "pop-up" que apareix al clicar el botó. La nova norma creada s'assigna a la gramàtica seleccionada i es grava correctament a la carpeta corresponent a la gramàtica. En cas de que aquesta carpeta no existeixi es crea de nou.
- **Eliminar norma:** Elimina la forma seleccionada, tant de la llista de normes de la gramàtica com la instancia de l'objecte escriptable. Per evitar eliminar normes sense voler apareix un "pop-up" a l'usuari preguntant si realment vol eliminar-la.
- **Duplicar norma:** Crea una norma nova copiant tots els paràmetres de la norma seleccionada.

- **Crear forma:** Crea un *prefab* nou a la carpeta de formes de la gramàtica seleccionada. Si aquesta carpeta no existeix es crea de nou. La forma creada pot ser cubica, plana o buida. Al crear-la hi ha un camp per definir el símbol de la forma, que servirà també per donar nom al fitxer del *prefab*.
- Finalment es troba un desplegable que permet seleccionar el tipus de normes entre normes de producció i normes de post-producció. Depenent del mode seleccionat, al selector de normes només apareixeran unes normes o les altres per facilitar el procés, ja que son fases diferents de la definició d'una gramàtica.

La resta de la interfície està dividida en dos blocs verticalment, a la dreta trobem el selector de normes, i a l'esquerra l'inspector de normes.

El selector de normes mostra en una quadrícula les normes presents en la gramàtica seleccionada i l'usuari ha de clicar la que vulgui editar. Per facilitar la busca de normes s'ha incorporat una barra de busca per filtrar les normes per nom.

Finalment, l'inspector de normes mostra tots els paràmetres que l'usuari pot editar en la norma seleccionada. A diferència de l'inspector de Unity, només es mostren els paràmetres rellevants i d'una manera més compacta per facilitar la lectura.

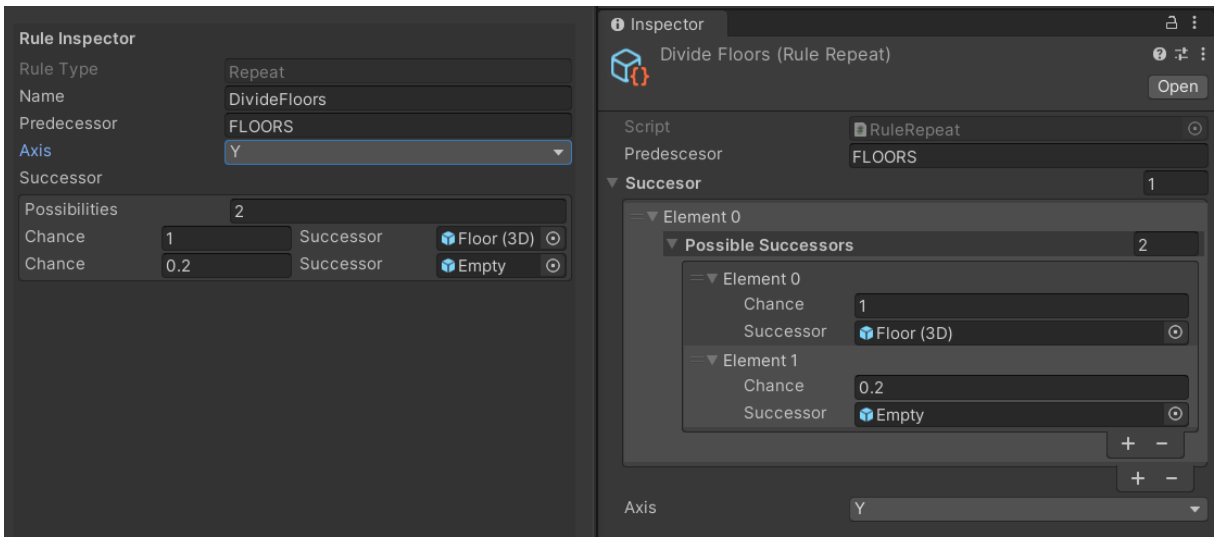


Figura 6-10. Comparació d'inspectors. A la dreta l'inspector de Unity. A la esquerra l'inspector propi.

A més a més, tots els paràmetres contenen "tooltips" (menús flotants que apareixen al passar el ratolí per sobre del nom del paràmetre) que contenen informació per guiar l'usuari, i alguns

dels paràmetres despleguen missatges d'ajuda si el valor introduït pot donar problemes en la execució de l'algorisme.

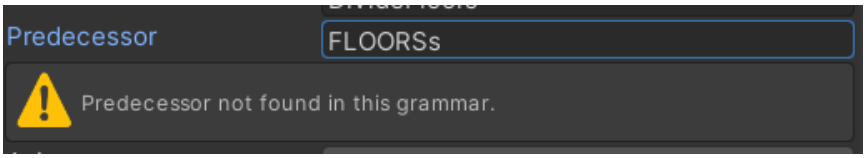


Figura 6-11. Exemple de missatge d'ajuda. En aquest cas, el símbol del predecessor no coincideix amb cap forma present a la gramàtica.

6.3.6.2. Constructor de formes terminals

El constructor de formes terminals és una eina pensada per preparar les *assets* finals per ser utilitzades en el procés de post-producció. La interfície d'aquesta eina conté cinc passos simples perquè a l'usuari li sigui fàcil. La interfície està composta per tres elements: a dalt de tot els botons per tirar endavant i endarrere els passos i l'indicador del pas en el que es situa, després una breu descripció del que ha de fer l'usuari i finalment els paràmetres que l'usuari ha de modificar.

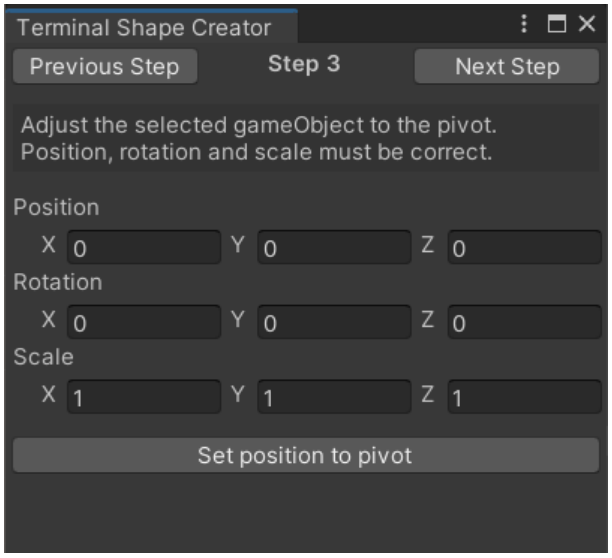


Figura 6-12. Interfície del constructor de formes terminals.

En el primer pas l'usuari selecciona la zona de treball seleccionant una posició en el món, que es marca en l'escena amb un objecte i determinarà l'arrel o punt de pivot de l'objecte final. En el segon pas l'usuari selecciona el model 3D amb el que vol treballar, el qual ha d'estar en l'escena. En el tercer pas l'usuari ajusta la posició, rotació i escala perquè s'ajusti als valors que ha de tindre la forma final, tenint en compte la posició i orientació de l'arrel. En el quart pas l'usuari defineix el volum de malla utilitzant un indicador cúbic. En el cinquè pas es defineix

el nom que tindrà l'objecte final i en l'ultim pas s'ajusta el model 3d perquè tingui l'arrel en el la posició marcada i una escala de una unitat en els tres eixos, d'aquesta manera es pot ajustar l'escala fàcilment durant la post-producció.

6.3.6.3. Constructor

El constructor o "Builder" és la interfície més senzilla, aquesta té tres paràmetres per definir les formes que defineixen la configuració inicial, la gramàtica que s'utilitzarà i un booleà per indicar si s'ha d'executar o no el procés de post-producció.

El següent segment de la interfície és el control de la llavor del generador de nombres aleatoris de Unity. L'usuari pot triar si vol que cada vegada es generi amb una llavor diferent, bloquejar-la o introduir una llavor seleccionada anteriorment.

Finalment, conté un botó per executar la derivació. Quan s'executa la derivació es creen dos objectes nous anomenats "Derivation" que conté l'arbre de derivació, i "Building" que conté tots els models 3D. El nom de l'objecte "Derivation" sempre va acompanyat amb la llavor que s'ha utilitzat per aconseguir el resultat.

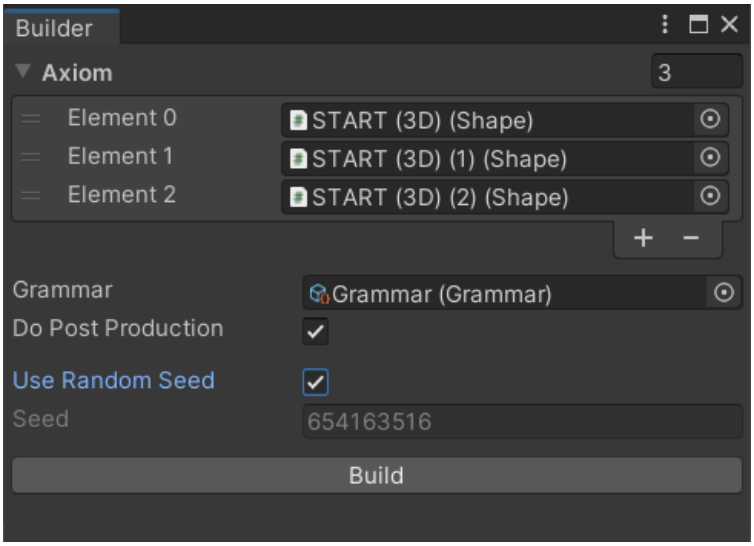


Figura 6-13. Interfície del constructor.

7. Resultats

El resultat final del desenvolupament és un sistema capaç de treballar amb una gran quantitat de representacions de coneixement, ja que depenent del detall de generació de la gramàtica pot adaptar-se des del fragments experiencials i construir un nivell o fins i tot generar fins a l'últim dels detalls més petits, tot depèn de la quantitat de normes que conformin la gramàtica.

El model desenvolupat genera espais, a partir de formes terminals que poden ser fragments o altres espais, és un sistema *offline* i determinista, controlat per una llavor. Per la manca de sensibilitat al context d'aquesta gramàtica el model tendeix a ser de generar i provar, degut a que, depenent de la gramàtica, pot ser que al model li costi generar resultats satisfactoris. El model és de generació automàtica, ja la interacció de l'usuari amb el contingut generat és limitada.

Utilitzant el model generador es poden distingir dos tipus de resultats depenent de la construcció de la gramàtica. Per una banda hi ha les gramàtiques que es comporten de manera reduccionista, que son els que l'àmbit total del resultat és igual o menor a l'àmbit de l'axioma. Aquestes gramàtiques es comporten més com si fos un escultor tallant un bloc de pedra i es basa principalment en tallar la forma inicial d'una manera especifica.

Per altra banda hi ha gramàtiques que es comporten de manera expansionista. Fent un us ampli de les normes d'àmbit, aquestes gramàtiques mouen les noves formes creades fora de l'àmbit inicial. Fent ús de bucles de normes poden fer generacions quasi infinites, si no se li dona cap camí de sortida per aturar el bucle.

A continuació es mostren dos exemples de edificacions creades amb el *framework* desenvolupat, una és de tipus reduccionista i l'altra de tipus expansionista.

7.1. Mansió Deteriorada

En aquest exemple, decorat utilitzant *assets* del paquet Flooded Grounds (Sandro T, 2019)[2], és un exemple de gramàtica reduccionista degut a que l'àmbit final de l'edifici està definit per l'àmbit de la forma inicial.

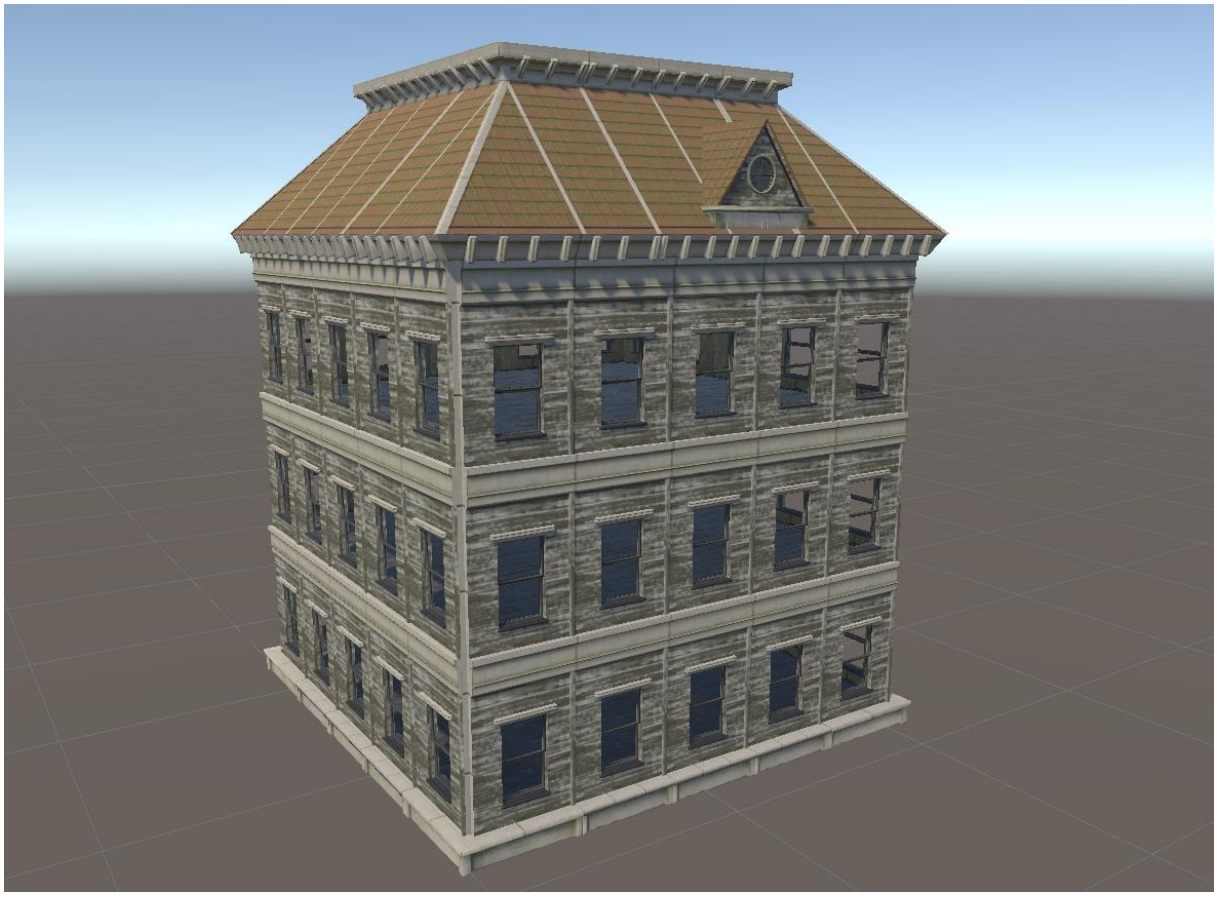


Figura 7-1. Exemple de generació utilitzant la gramàtica "Mansió deteriorada".

La gramàtica en qüestió defineix tres espais diferenciats per generar aquest tipus de mansions: la teulada, el cos de l'edifici i els fonaments. Cada una d'aquestes parts és tractada d'una manera diferent. Per exemple, a la teulada se li aplica una norma de marge per definir tot el perímetre de les aigües, mentre que al cos de l'edifici se li aplica una forma de repetició per definir els pisos.

[2] Sandro T (2019). Flooded Grounds. *Unity Asset Store*. Recollit de https://assetstore.unity.com/packages/3d/environments/flooded-grounds-48529

Aquesta gramàtica és bastant senzilla i no conté gaires normes. El detall en aquest model es pot donar de dues maneres, o bé utilitzant models 3D ben decorats, o bé afegint més normes per perfilar tots els detalls de la estructura. En el cas d'aquest exemple no es dona cap dels dos casos i és per això que la estructura generada pot semblar plana o repetitiva.

```
START → Split("Y", {7}, "ToRoot") {ROOF | BODY}
ROOF → Margin("Y", 7 , "Absolute", "FacingOut", True) {ROOFCORNER | ROOFEDGE | ROOFCENTER}
ROOFEDGE → Repeat("X") {[15:ROOFEDGESEGMENT][1:ROOFSEGMENTWINDOW]}
ROOFCENTER → Component("Y", "Top") {NULL | ROOFTOP | NULL}
ROOFTOP → Scope(T:(-3.50, -3.50, 0.00), "Add", "Self"; S:(7.00, 7.00, 0.00), "Add",
"Absolute"){ROOFTOP2}
ROOFTOP2 → Grid("Y"){ROOFTOPSEGMENT}
BODY → Split("Y", {1}, "FromRoot") {FOOTING | FLOORS}
FOOTING → Margin("Y", 4 , "Absolute", "FacingOut", False) {FOOTINGCORNER | FOOTINGEDGE}
FOOTINGEDGE → Repeat("X") {FOOTINGEDGESEGMENT}
FLOORS → Repeat("Y") {FLOOR}
FLOOR → Component("Y", "SidesPlusBottom") {FACADE | NULL | WOODFLOOR}
FACADE → Repeat("X") {FACADESEGMENT}
WOODFLOOR → Scope(S:(0.00, 0.00, -0.85), "Add", "Absolute"){WOODFLOOR2}
WOODFLOOR2 → Grid("Z"){WOODFLOORSEGMENT}
```

Figura 7-2. Normes de la gramàtica de la mansió.

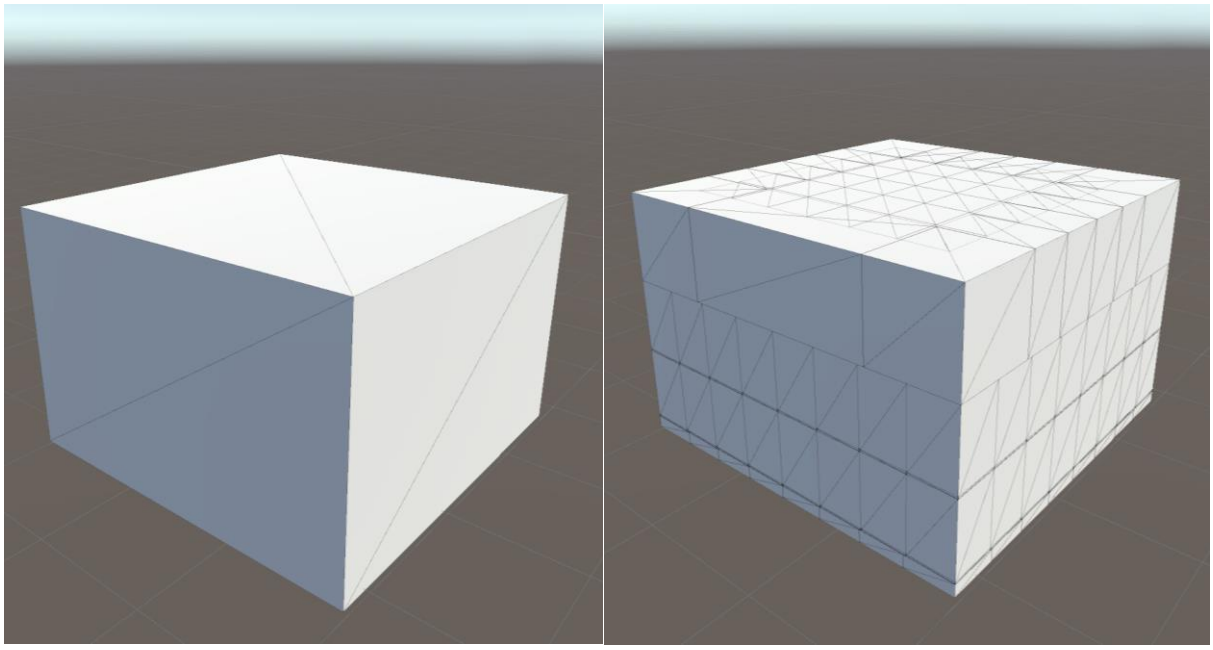


Figura 7-3. Procés de derivació. A la esquerra la forma inicial. A la dreta la representació visual de la derivació completada.

En la Figura 7-3 es pot apreciar que l'àmbit total de la forma inicial i del resultat de la generació es manté invariat. Tot i que en el procés de producció els models 3D donen una mica de volum a les parets, aquestes segueixen sense canviar gaire.

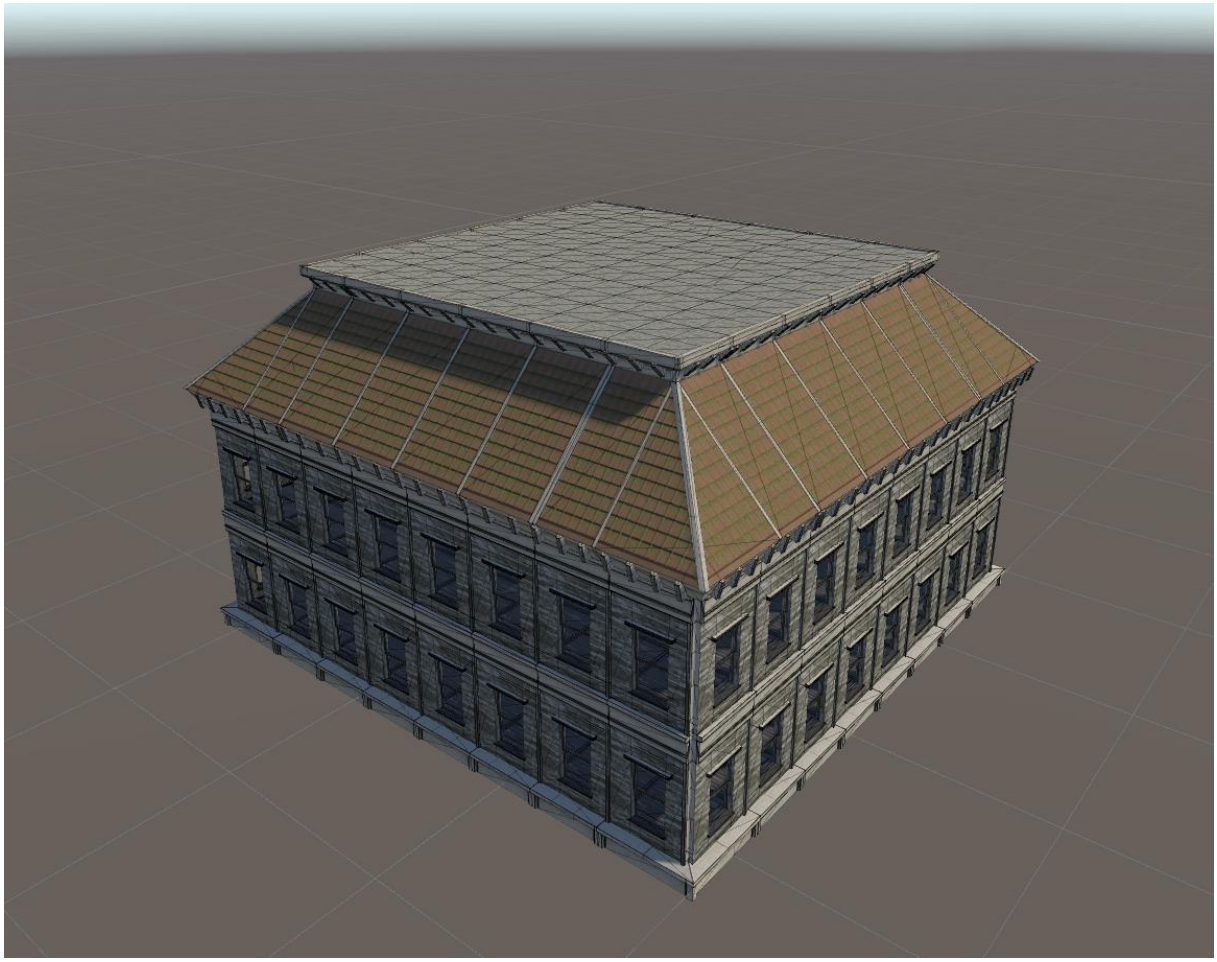


Figura 7-4. Post-producció de l'exemple de la figura 7-3.

7.2. Pagoda

El següent exemple, decorat utilitzant les *assets* del paquet Chinese Modular House (Erbelio 3D, 2022)[3], es genera una pagoda a partir del primer pis, és a dir, la forma inicial defineix les dimensions de la habitació de la planta baixa.

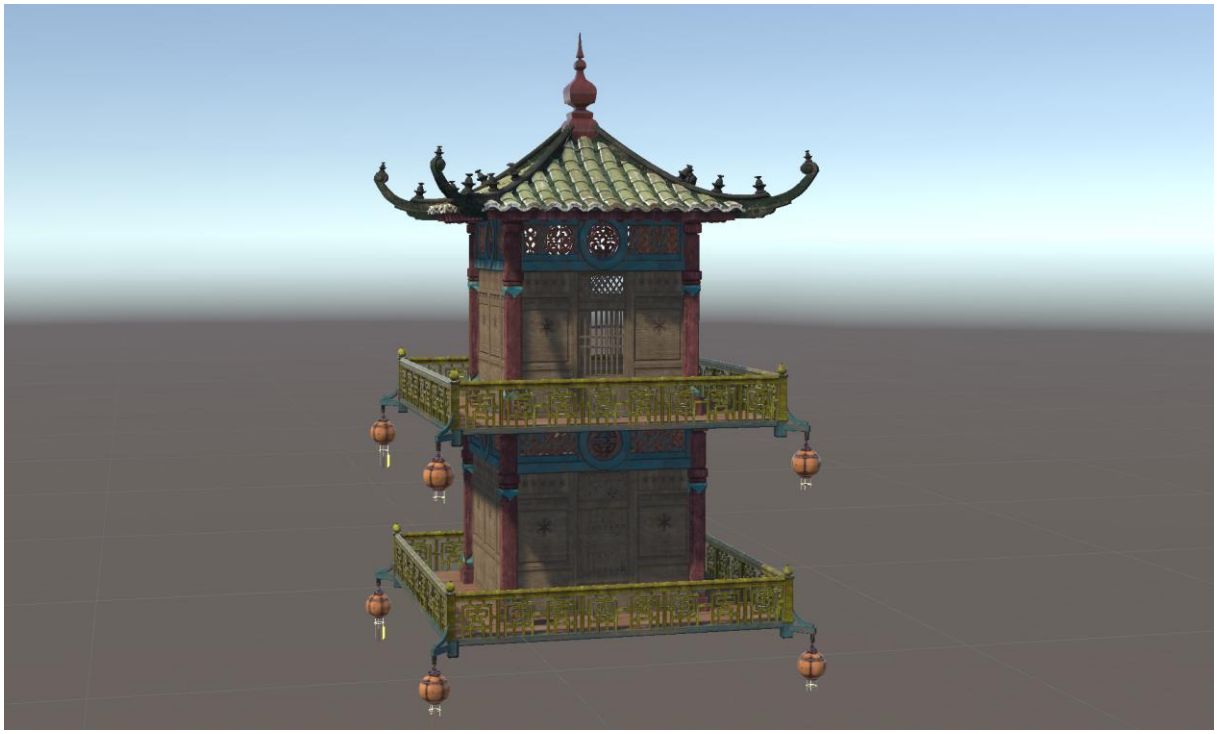


Figura 7-5. Exemple de generació utilitzant la gramàtica "Pagoda".

Aquest és un exemple de generació expansionista, ja que el resultat final sempre excedeix per molt l'àmbit de la forma inicial. Com es pot apreciar en les normes de producció d'aquesta gramàtica (Figura 7-6), trobem un us més gran de normes de tipus d'àmbit. Això permet a la gramàtica desplaçar les formes creades fora de l'àmbit inicial i per tant créixer. En aquest cas concret la norma de tipus component aplicada a la forma "Room" divideix la sala en les parts importants: el terra s'expandeix per formar els balcons, les parets es generen incorporant decisions pseudo-aleatòries per generar varietat i finalment el sostre de la habitació és la peça important. El sostre pot ser que es converteixi en teulada i finalitzi el procés de derivació o bé

[3] Erbelio 3D (2022). Erbeilo 3D - Built_in RP Chinese Moduler House. *Unity Asset Store.* Recollit de https://assetstore.unity.com/packages/3d/environments/erbeilo-3d-built-in-rp-chinese-moduler-house-217380

que generi una altra forma de tipus “Room” a sobre seu, i per tant, que es torni a repetir el procés. Aquesta forma d’aplicar les normes en bucle pot ser perillosa si no existeix un camí de sortida, ja que bloquejarà el programa fins que exhaureixi la memòria si no acaba el bucle.

```
START → Margin("Y", 0.15 , "Absolute", "FacingOut", True) {CornerColumn | EMPTY | Room}

CornerColumn → Scope(T:(-0.15, 0.00, -0.15), "Add", "Self"; S:(0.50, -1.00, 0.50), "Set",
"Absolute"){Column}

Room → Component("Y", "AllFaces") {Wall | Cieling | Floor}

Wall → Scope(T:(0.00, 0.00, -0.10), "Add", "Self"; S:(0.00, 0.00, 0.15), "Add",
"Absolute"){Wall2}

Wall2 → Split("Y", {1, 0, 2}, "ToRoot") {WallTop | WallBeam | WallDoors}

WallBeam → Scope(T:(-0.10, -0.10, -0.10), "Add", "Self"; S:(0.10, 0.10, 0.10), "Add",
"Absolute"){Beam}

WallDoors → Repeat("X") {WallSegment}

WallSegment → Scope(){[50:WallFull][50:WallGrid]}

Floor → Scope(T:(-2.00, -2.00, -0.10), "Add", "Self"; S:(4.00, 4.00, -0.85), "Add",
"Absolute"){Floor2}

Floor2 → Margin("Z", 0.2 , "Absolute", "FacingOut", True) {RailingCorner | RailingBase |
Floor3}

Floor3 → Grid("Z"){FloorSegment}

RailingCorner → Scope(S:(0.00, 0.00, 1.25), "Add", "Absolute"){RailingCorner2}

RailingCorner2 → Split("Z", {0, 15}, "FromRoot") {LampPost | RailingCorner3}

LampPost → Scope(S:(1.00, 1.00, 1.00), "Set", "Absolute"){LampPost2}

RailingBase → Scope(S:(0.00, 0.00, 1.10), "Add", "Absolute"){RailingEdge}

RailingEdge → Split("Z", {0, 15}, "FromRoot") {Beam | Railing}

Railing → Repeat("X") {RailingSegment}

Cieling → Scope(){[87.50:ToNewFloor][12.50:ToRoof]}

ToRoof → Scope(T:(0.00, 0.75, 0.00), "Add", "World"; S:(0.00, 0.00, 0.75), "Add",
"Absolute"){Roof}

ToNewFloor → Scope(T:(0.00, 5.00, 0.00), "Add", "World"; S:(0.00, 0.00, 5.00), "Add",
"Absolute"){NewFloor}

NewFloor → Scope(T:(0.00, 0.00, 5.00), "Add", "Self"; R:(-90.00, 0.00, 0.00), "Add", "World";
){NewFloor2}

NewFloor2 → Margin("Y", 0.15 , "Absolute", "FacingOut", True) {CornerColumn | EMPTY | Room}
```

Figura 7-6. Normes de producció de la gramàtica "Pagoda"

En la figura 7-7 es pot apreciar clarament la tendència expansionista d’aquesta gramàtica, i també la quantitat de detall d’aquesta gramàtica si la comparem amb l’exemple anterior. La gramàtica de la mansió no aporta pràcticament cap detall en la derivació i ho fa tot en el procés de post-producció mentre que en la gramàtica de la pagoda es pot diferenciar clarament l’estructura abans del procés de post-producció.

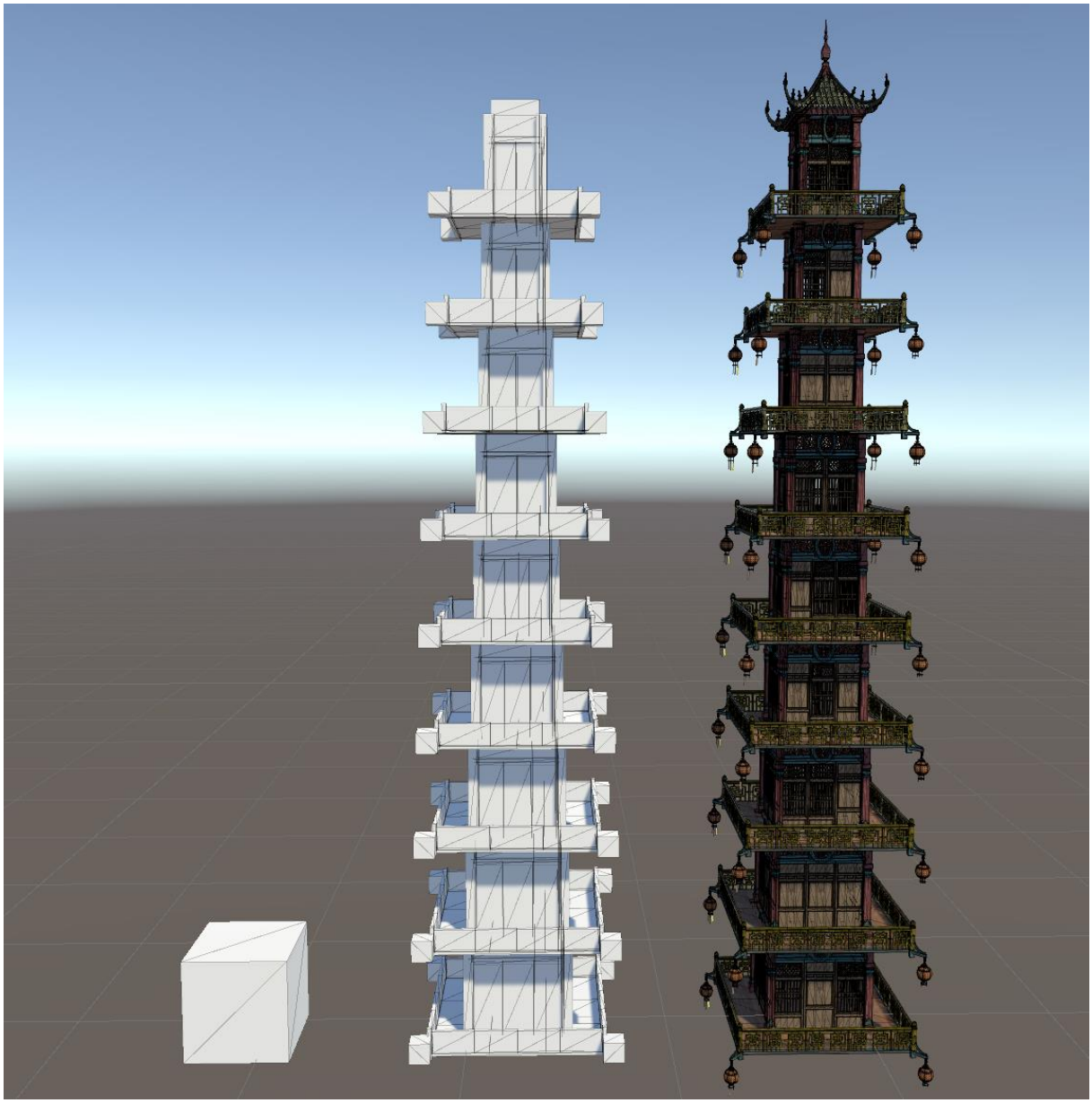


Figura 7-7. Els passos de la generació d'esquerra a dreta. A l'esquerra la forma inicial, al centre la visualització de la derivació completada i a la dreta el resultat final.

7.3. Línies de millora

Durant el desenvolupament de la eina s'han fet notar certs aspectes que, tot i no haver-se implementat per la brevetat de la fase de desenvolupament, millorarien les capacitats i usabilitat del model de generació.

7.3.1. Variables en els paràmetres.

En el sistema desenvolupat tots els paràmetres de les normes son fixes i no varien durant el procés de derivació. Això provoca que durant el procés de derivació cada norma s'aplica sense tenir cap informació del context. Una gramàtica que és sensible al context permet generar resultats més coherents.

Per altra banda en gramàtiques circulars, on les normes fan un bucle, són perilloses sense variables de control. Si la única manera de sortir de un bucle és pseudo-aleatòria, pot ser que el bucle continuï fins a exhaurir la memòria.

7.3.2. Capes de derivació.

Les capes de derivació és fer el procés de derivació en múltiples passos, és a dir, tenir varies gramàtiques on el resultat de una és l'axioma de la següent. Aquesta característica permet a l'usuari més control sobre l'ordre en el que s'apliquen les normes, ja que en el sistema actual aquest ordre no es pot controlar i per la naturalesa recursiva de l'algorisme l'ordre depèn de la profunditat en l'arbre de derivació.

7.3.3. Snapping i Oclusió.

Així com ho implementa les gramàtiques CGA de Müller et. al. (2006) l'*snapping* i la oclusió son una fantàstica eina per millorar els resultats, ja que permet a la gramàtica treballar amb axiomes complexos i fer que el resultat final sigui coherent en totes les línies. La oclusió també ajuda amb la optimització, ja que elimina totes aquelles formes que no son visibles en el resultat final.

7.3.4. Millores en l'editor de gramàtiques

Per poder facilitar el moviment de gramàtiques entre projectes seria útil un sistema de exportació i importació utilitzant fitxers .unitypackage o serialitzar totes les normes i guardar les gramàtiques en únic fitxer .json. En el sistema desenvolupat l'única manera de transferir

gramàtiques entre projectes és copiar la carpeta des de l'explorador de Windows. Un sistema que, tot i que funciona perfectament per les normes de producció i les formes, falla al copiar les normes de post-producció. Aquest error és degut a que les normes de post-producció tenen referències a *prefabs* i aquests tenen un seguit de dependències de models i textures que també s'han de transferir. Un sistema de exportació i importació també permetria fusionar gramàtiques i per tant poder desenvolupar parts d'una gramàtica per separat i després ajuntar-les.

Per poder accelerar el procés de creació de normes i depurar els problemes fàcilment, seria interesant la opció de que les normes poguessin ser activades i desactivades. Així l'usuari podria tallar el procés de derivació sense la necessitat de modificar o eliminar cap norma.

7.3.5. Millora de la norma d'àmbit

La norma d'àmbit del sistema actual és poc usable degut a que l'usuari no pot triar l'ordre en el que s'apliquen les transformacions, sempre es fa primer la translació, després la rotació i finalment la escala. Per solucionar aquest problema és possible crear sub-normes d'àmbit que es centrin només en la translació, rotació o escala, així com afegir-ne alguna més com ara la simetria.

Aquestes sub-normes les afegiria l'usuari en una llista reordenable de Unity per tal de poder triar l'ordre en el que s'aplicaran i així poder crear algorismes de transformació.

7.4. Nom de la eina

Tot i que la eina no s'ha pogut publicar encara a la Asset Store de Unity, tal i com marcava un dels objectius secundaris, si que s'ha treballat l'aspecte de màrqueting i s'ha escollit el nom de Grammar Based Building Generator per a la eina. Amb les sigles GBBG per ser més ràpid de pronunciar.

8. Conclusió

Com s'ha demostrat en els punts de desenvolupament i resultats, l'eina creada ha assolit els objectius principals que es van plantejar a l'inici del treball, però no els objectius secundaris. El fet de no haver pogut assolir cap objectiu secundari porta a pensar que l'abast del projecte era més gran de l'esperat. També fomenta aquesta idea la quantitat de línies de millora que s'han presentat amb els resultats per tal de que la eina sigui més robusta, ja que l'estat actual de la eina es pot considerar més una beta que no pas una versió final.

Tot i així es pot remarcar que els objectius principals s'han assolit, però no amb el nivell de polit que s'esperava a l'inici del projecte.

L'àmbit teòric del desenvolupament, és a dir, la definició formal de la gramàtica que s'utilitzarà ha estat la part més senzilla del projecte, ja que s'han estudiat varis tipus de gramàtiques de formes i s'han seleccionat les característiques més optimes de cadascuna. Per altra banda, en l'àmbit de implementació no ha estat tan senzill. La versió implementada de la gramàtica només pot treballar amb formes cubiques o amb quads, mantenint tots els angles rectes. Aquesta és una decisió que es va prendre a l'inici del desenvolupament ja que les proves realitzades durant la fase de investigació van concloure que treballar amb qualsevol tipus de forma faria que la implementació de totes les normes fos molt més lenta i per tant l'abast del projecte seria massa gran.

El principal contratemps del projecte ha sigut la creació de interfícies, ja que en la fase de investigació no es va estudiar les tecnologies que ofereix Unity i el les interficies s'han desenvolupat utilitzant IMGUI, que en comparació a UI Toolkit requereix un coneixement més profund del sistema per poder treure'n tot el suc. La documentació de la eina creada no ha suposat cap contratemps i descriu adequadament tots els aspectes de la eina perquè altres persones puguin usar-la.

Per acabar amb els objectius, els objectius secundaris no s'han assolit per la falta de temps de desenvolupament, la exportació de models FBX no suposa gaire treball, però per ser fàcilment usable s'hauria d'acompanyar d'una altra interfície. Per publicar la eina a la Asset Store de Unity fa falta més polit general i implementar algunes de les propostes de millora del punt 7.3.

El cronograma del projecte s'ha seguit mitjanament, la investigació i la elaboració de l'avantprojecte van seguir-lo sense complicacions. Per altra banda el desenvolupament ha tingut alts i baixos. Es va haver d'aturar durant una part del març i això ha fet que s'hagi hagut d'allargar fins al juny i per tant haver de redactar la memòria sense haver acabat el desenvolupament.

En relació a eines similars, GBBG és la única que implementa un sistema de gramàtiques per generar edificis, el que més s'hi assembla és Procedural Generation Grid (FImpossible Creations, 2022), que tot i tindre més característiques es centre en la generació d'espais interiors únicament. Per altra banda l'ús de gramàtiques per la generació procedimental està de capa caiguda, ja que tota la investigació del camp de la generació procedimental s'està centrant actualment en models generatius basats en IA. Aquests models, tot i que no s'han començat a implementar en jocs comercials, mostren un progrés de desenvolupament molt accelerat, i per tant pot ser que en molt poc temps comencin a substituir models més "tradicionals" com ara les gramàtiques.

En conclusió, el projecte ha aconseguit els seus objectius primaris, i malgrat no haver assolit els objectius secundaris a causa de retards en el desenvolupament, l'eina desenvolupada ha sigut capaç de generar varis tipus d'estructures i disposa de tot el contingut principal que es volia implementar.

9. Referencies

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis. Espoo2002.* VTT Publications 478.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys, 23*(3), 345-405. Recollit de https://doi.org/10.1145/116873.116880

Barriga, N. A. (2019). A Short Introduction to Procedural Content Generation Algorithms for Videogames. *International Journal on Artificial Intelligence Tools, 28*(2). Recollit de https://doi.org/10.1142/S0218213019300011

Bartle, R. (2003). *Designing Virtual Worlds.* Recollit de https://www.researchgate.net/publication/200025892_Designing_Virtual_Worlds

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., . . . Thomas, D. (2001). *Manifest per al desenvolupament àgil de programari.* Recollit de agilemanifesto.org: https://agilemanifesto.org/iso/ca/manifesto.html

Booth, M. (2009). The AI Systems of Left 4 Dead. *Artificial Intelligence and Interactive Digital Entertainment Conference.* Stanford.

Brathwaite, B., & Schreiber, I. (2008). *Challenges for Game Designers* (1 ed.). Charles River Media, Inc.

Browne, C., & Marie, F. (2010). Evolutionary Game Design. *IEEE Transactions on Computational Intelligence and AI in Games, 2*, 1-16. doi:10.1109/TCIAIG.2010.2041928

Browne, D. (2007). Yavalath. Nestorgames.

Chen, G., Esch, G., Wonka, P., Müller, P., & Zhang, E. (2008). Interactive Procedural Street Modeling. *SIGGRAPH '08: ACM SIGGRAPH 2008 papers. 103*, p. 1-10. Los Angeles, California: Association for Computing Machinery. Recollit de https://doi.org/10.1145/1399504.1360702

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory, 2*(3), 113-124.

Chomsky, N. (1957). *Syntactic Structures.* Mouton & Co. Publishers, The Hague.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control, 2*, 137-167.

Chomsky, N. (1963). Formal properties of grammars. A R. D. Luce, R. R. Bush, & E. Galanter (Ed.), *Handbook of Mathematical Psychology* (Vol. 2, p. 323-418). Nova York i Londres: John Wiley and Sons, Inc.

Chopard, B., & Droz, M. (1998). *Cellular Automata Modeling of Physical Systems.* doi:10.1007/978-0-387-30440-3_57

Colton, S. (2002). Automated puzzle generation. *Proceedings of the AISB'02 Symposium on AI and Creativity in the Arts and Science.* Citeseer.

de Berg, M., Cheong, O., van Kreveld, M., & Overmars, M. (2008). Voronoi Diagrams. A *Computational Geometry* (p. 147-171). Springer. Recollit de https://doi.org/10.1007/978-3-540-77974-2_7

Dhariwal, P., Jun, H., Payne, C., Kim, J. W., Radford, A., & Sutskever, I. (2020). Jukebox: A Generative Model for Music. Recollit de https://doi.org/10.48550/arxiv.2005.00341

FImpossible Creations. (2022). Procedural Generation Grid (Beta). *Unity Asset Store*. Recollit de https://assetstore.unity.com/packages/tools/utilities/procedural-generation-grid-beta-195535

Gao , J., Shen , T., Wang , Z., Chen , W., Yin , K., Li , D., . . . Fidler, S. (2022). GET3D: A Generative Model of High Quality 3D Textured Shapes Learned from Images. *Advances In Neural Information Processing Systems.*

Garfield, R. (2000). Metagames. A J. Dietz (Ed.), *Horsemen of the Apocalypse: Essays on Roleplaying.* Jolly Roger Games.

Grup Enciclopèdia. (1996). Gramàtica. A *Gran Enciclopèdia Catalana.* Barcelona: Grup Enciclopèdia. Recollit de https://www.enciclopedia.cat/gran-enciclopedia-catalana/gramatica-0

Gumin, M. (21 / 07 / 2022). *Wave Function Collapse*. Recollit de GitHub: https://github.com/mxgmn/WaveFunctionCollapse

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation* (2 ed.). Prentice Hall PTR. doi:10.5555/521706

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013). Procedural content generation for games: A survey. *ACM Transactions on Multimedia Computing, Communications, and Applications, 9*(1), 1-22. Recollit de https://doi.org/10.1145/2422956.2422957

Hunter, T. (2021). The Chomsky Hierarchy. A N. Allott, T. Lohndal, & G. Rey (Ed.), *Companion to Chomsky* (1 ed., p. 74-95). John Wiley & Sons, Inc. Recollit de https://doi.org/10.1002/9781119598732.ch5

Larive, M., & Gaildrat, V. (2006). Wall Grammar for Building Generation. *GRAPHITE '06: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia* (p. 429–437). Kuala Lumpur, Malaysia: Association for Computing Machinery. Recollit de https://doi.org/10.1145/1174429.1174501

Lindenmayer, A. (Març / 1968). Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of Theoretical Biology, 18*(3), 280-299. Recollit de https://doi.org/10.1016/0022-5193(68)90079-9

Liu , J., Snodgrass , S., Khalifa , A., Risi , S., Yannakakis , G. N., & Togelius , J. (2021). Deep learning for procedural content generation. *Neural Computing and Applications, 33*, 19-37. Recollit de https://doi.org/10.1007/s00521-020-05383-8

Lorensen, W. E., & Cline, H. E. (1987). Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *ACM SIGGRAPH Computer Graphics, 21*(4), 163-169. Recollit de https://doi.org/10.1145/37402.37422

Müller, P., Wonka, P., Haegler, S., Ulmer, A., & Van Gool, L. (2006). Procedural modeling of buildings. *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers* (p. 614-623). Boston, Massachusetts: Association for Computing Machinery. Recollit de https://doi.org/10.1145/1179352.1141931

Nemesys9999. (10 / 04 / 2022). !!! DO NOT BUY THIS GAME - N'ACHETEZ PAS CE JEU !!! *Steam*. Recollit de https://steamcommunity.com/profiles/76561198206990592/recommended/1525620/

Nitsche, M. (2008). *Video Game Spaces: Image, Play, and Structure in 3D Worlds.* The MIT Press.

Norman, D. A. (2002). The design of everyday things Basic Books. *New York: Perseus*.

Parish, Y. I., & Müller, P. (2001). Procedural Modeling of Cities. *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (p. 301–308). Association for Computing Machinery. Recollit de https://doi.org/10.1145/383259.383292

Peng, X. B., Guo, Y., Halper, L., Levine, S., & Fidler, S. (2022). ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters. *ACM Transactions on Graphics, 41*(4). Recollit de https://doi.org/10.1145/3528223.3530110

Pi, X., Song, J., Zeng, L., & Li, S. (2006). Procedural Terrain Detail Based on Patch-LOD Algorithm. A Z. Pan, R. Aylett, H. Diener, X. Jin, S. Göbel, & L. Li (Ed.), *Edutainment'06: Proceedings of the First international conference on Technologies for E-Learning and Digital Entertainment* (p. 913-920). Hangzhou, China: Springer. Recollit de https://doi.org/10.1007/11736639_111

Prusinkiewicz, P., & Hanan, J. (1989). Lindenmayer Systems, Fractals, and Plants. A *Lecture Notes in Biomathematics* (Vol. 79). Springer.

Prusinkiewicz, P., & Lindenmayer, A. (1990). *The Algorithmic Beauty of Plants.* Springer.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., . . . Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. Recollit de https://doi.org/10.48550/arxiv.2102.12092

Rist, I., & McGinnigle, J. (sense data). *What is a Tensor?* (P. Midgley, & E. Bithell, Ed.) Recollit de Dissemination of IT for the Promotion of Materials Science. University of Cambridge: https://www.doitpoms.ac.uk/tlplib/tensors/what_is_tensor.php

Shaker, N., Togelius, J., & Nelson, M. J. (2016). *Procedural Content Generation in Games.* Springer. Recollit de https://doi.org/10.1007/978-3-319-42716-4

Smelik, R., Tutenel, T., Kraker, K., & Bidarra, R. (2010). Integrating Procedural Generation and Manual Editing of Virtual Worlds. *PCGames '10: Proceedings of the 2010 Workshop on Procedural Content Generation in Games* (p. 1-8). Monterey, California: Association for Computing Machinery. Recollit de https://doi.org/10.1145/1814256.1814258

Smith, G. (2015). Procedural Content Generation: An Overview. A S. Rabin (Ed.), *Game AI Pro 2* (p. 501-518). CRC Press. Recollit de http://www.gameaipro.com/

Smith, G., Whitehead, J., & Mateas, M. (2010). Tanagra: A mixed-initiative level design tool. *Proceedings of the Fifth International Conference on the Foundations of Digital Games* (p. 209-216). Monterey, California: Association for Computing Machinery. Recollit de https://doi.org/10.1145/1822348.1822376

Stålberg, O. (2018). Wave Function Collapse in Bad North. *Everything Procedural Conference 2018.* Breda. Recollit de https://www.youtube.com/watch?v=0bcZb-SsnrA

Stålberg, O. (2019). Organic Towns from Square Tiles. *INDIECADE EUROPE 2019*. Paris, França. Recollit de https://www.youtube.com/watch?v=1hqt8JkYRdI

Stiny, G. (1975). *Pictorial and Formal Aspects of Shape and Shape Grammars. Interdisciplinary Systems Research.* Bitkhäuser, Basel. Recollit de https://doi.org/10.1007/978-3-0348-6879-2

Stiny, G. (1980). Introduction to Shape and Shape Grammars. *Environment and planning B: planning and design, 7*(3), 343-351. Recollit de https://doi.org/10.1068/b070343

Stiny, G. (1982). Spatials Relations and Grammars. *Enviroment and Plannig B, 9*(1), 113-114. doi:10.1068/b090113

Togelius, J., Kastbjerg, E., Schedl, D., & Yannakakis, G. N. (2011). What is Procedural Content Generation? Mario on the Borderline. *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games.* Bordeaux, France: Association for Computing Machinery. Recollit de https://doi.org/10.1145/2000919.2000922

Togelius, J., Yannakakis, G. N., Stanley, K. O., & Browne, C. (2010). Search-Based Procedural Content Generation. A C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. I. Esparcia-Alcazar, C.-K. Goh, . . . G. N. Yannakakis (Ed.), *Applications of Evolutionary Computation. EvoApplications 2010. Lecture Notes in Computer Science. 6024*, p. 141-150. Berlin, Heidelberg: Springer. Recollit de https://doi.org/10.1007/978-3-642-12239-2_15

Van Verth, J. M., & Bishop, L. M. (2008). *Essential mathematics for games and interactive applications: A programmer's guide* (2ª ed.). Elsevier.

Vieira, R., Dembogurski, B., Alvim, L., & Braida, F. (2018). A Cognitive Architecture for Agent-Based Artificial Life Simulation. *Computational Science and Its Applications – ICCSA 2018. ICCSA 2018. Lecture Notes in Computer Science. 10960*, p. 197-213. Springer International Publishing.

Wonka, P., Wimmer, M., Sillion, F., & Ribarsky, W. (2003). Instant Architecture. *ACM Transactions on Graphics, 22*(3), 669667. Recollit de https://doi.org/10.1145/882262.882324

Yu, L., Cheng, Y., Sohn, K., Lezama, J., Zhang, H., Chang, H., . . . Jiang, L. (2022). MAGVIT: Masked Generative Video Transformer. arXiv. Recollit de https://doi.org/10.48550/arxiv.2212.05199

9.1. Programari

ArcGis CityEngine (Versió 2022.1) [Editor 3D]. (2008). Esri R&D Center Zurich.

Borderlands (Versió per a ordinador) [Videojoc]. (2009). Gearbox Software.

BorisTheBrave (2022). Tessera (Versió 5.2.0). [Eina de Unity].

Building Crafter (Versió 0.8.5). [Eina de Unity]. (2020). 8Bit Goose Games.

Civilization (Versió per a ordinador) [Videojoc]. (1991). MicroProse.

Elite (Versió per a ordinador) [Videojoc]. (1984). AcornSoft.

Els Sims (Versió per a ordinador) [Videojoc]. (2000). Electronic Arts.

Forza Motorsport (Versió per a ordinador) [Videojoc]. (2005). Turn 10 Studios.

Houdini (Versió 19.0.588) [Editor 3D]. (1996). Side Effects Software.

Left 4 Dead (Versió per a ordinador) [Videojoc]. (2008). Valve.

Maya (Versió 2023.2) [Editor 3D]. (1998). Autodesk.

Minecraft (Versió per a ordinador) [Videojoc]. (2011). Mojang Studios.

Minecraft Dungeons (Versió per a ordinador) [Videojoc]. (2020). Mojang Studios.

Moeglich, F. (2022). Procedural Generation Grid (Versió 1.6.1) [Eina de Unity]. Fimpossible Creations.

ProGen (Versió 1.0.0). [Eina de Unity]. (2020). (Browne D. , 2007) Dilmer Games.

Rogue (Versió per a ordinador) [Videojoc]. (1980). AI Design.

Slaves to Armok: God of Blood Chapter II: Dwarf Fortress (Versió per a ordinador) [Videojoc]. (2006). Bay 12 Games.

Spore (Versió per a ordinador) [Videojoc]. (2008). Maxis.

Temple Run (Versió per a mòbils) [Videojoc]. (2011). Imangi Studios.

The Architect: Paris (Versió per a ordinador) [Videojoc]. (2021). Enodo Games.

Townscaper (Versió per ordinador) [Videojoc]. (2021). Oskar Stålberg.

Ultima Online (Versió per a ordinador) [Videojoc]. (1997). Origin Systems.

Unity (Versió 21.3.8f1) [Motor de videojocs]. (2005). Unity Technologies.

Warframe (Versió per a ordinador) [Videojoc]. (2013). Digital Extremes.

10. Annex

GBBG Documentation