Generació procedural de coves en 3D

Grau en Enginyeria Informàtica

Computació

**Autor:**
Alec Lafita Secanilla

**Director:**
Alejandro Ríos Jerez,
departament de Computació

26 de juny de 2017

Universitat Politècnica de Catalunya (UPC)
– BarcelonaTech

Facultat d’Enginyeria Informàtica (FIB)
Resum

En aquest treball es pretén implementar un algoritme per poder dur a terme una generació procedural de coves, cavernes, mines o masmorres tridimensionals que el jugador podrà inspeccionar i recórrer per dins, així com ser una eina expansible per a poder integrar-ho al desenvolupament d’un videojoc.

Resumen

En este trabajo se pretende implementar un algoritmo para poder obtener una generación procedural de cuevas, cañones, minas o mazmorras tridimensionales que el jugador podrá inspeccionar e ir recorriendo por dentro, así como ser una herramienta expandible para poder integrarlo en el desarrollo de un videojuego.

Abstract

The aim of this project is to implement an algorithm to generate caves, caverns, mines and dungeons’ threedimensional models through procedural generation. This generation will give the player the possibility to explore it as well as to be an expandable tool that will help to integrate it on a videogame development.
Índex

1 Introducció i formulació del problema 8

2 Context 9
   2.1 Actors implicats ................................. 10
       2.1.1 Dissenyador, Programador, Tester .............. 10
       2.1.2 Director del projecte ........................... 11
       2.1.3 Desenvolupador extern com a usuari ............. 11
       2.1.4 Usuari final ................................. 11

3 Estat de l’art 12
   3.1 2D .................................................. 12
   3.2 Unió de mòduls .................................... 12
   3.3 Heightmap .......................................... 12
   3.4 L-system ........................................... 13
   3.5 Voxels i octrees .................................... 13
   3.6 Conclusions ....................................... 14

4 Objectius del projecte 16

5 Abast 17
   5.1 Possibles obstacles ............................... 18
       5.1.1 Errors al codi ................................. 18
       5.1.2 Durada del projecte ............................ 18
       5.1.3 Perdua del projecte ............................ 18
       5.1.4 Artifacts ................................. 19

6 Metodologia i rigor 20
   6.1 Metodologia ................................. 20
       6.1.1 Mida de l’equip ................................. 20
       6.1.2 Cicles de desenvolupament ....................... 20
       6.1.3 Feedback ................................. 20
       6.1.4 Metodologia final ............................... 20
   6.2 Eines de seguiment i validació ....................... 21
       6.2.1 Control de sprints ............................. 21
       6.2.2 Control de versions ............................. 21
   6.3 Eines i llenguatge de programació .......................... 21
7 Planificació temporal

7.1 Duració ................................................. 23
7.2 Pla de projecte ........................................... 23
  7.2.1 Preparació de l’entorn ................................. 23
  7.2.2 Fita inicial ........................................... 23
  7.2.3 Anàlisi i Implementació ............................... 24
  7.2.4 Fita final ............................................. 25
7.3 Temps de dedicació aproximat ............................. 25
7.4 Valoració d’alternatives .................................. 26
7.5 Recursos ...................................................... 26
  7.5.1 Humans ............................................... 26
  7.5.2 Hardware ............................................. 27
  7.5.3 Software ............................................. 27
7.6 Diagrama de Gantt ......................................... 28
7.7 Desviacions i conclusions .................................. 29

8 Gestió econòmica ............................................. 32

8.1 Costos de recursos humans ................................ 32
8.2 Costos de hardware ....................................... 32
8.3 Costos de software i llicències ............................ 33
8.4 Costos indirectes ......................................... 33
8.5 Pressupost total .......................................... 33
8.6 Control de gestió ......................................... 34
8.7 Desviacions .................................................. 34

9 Sostenibilitat .................................................. 36

9.1 Econòmica .................................................... 36
9.2 Social ......................................................... 37
9.3 Ambiental ..................................................... 38
9.4 Conclusions .................................................. 39

10 Identificació de lleis i regulacions ........................... 40

11 Justificació del projecte .................................... 41

11.1 Assignatures relacionades amb Computació ............. 41
11.2 Competències tècniques del projecte ..................... 41
16 No convexitat i no simplicitat ........................................... 54
17 Entrada de cova .......................................................... 54
18 Càlcul de l’àrea d’una polilínia en 2D ............................. 56
19 Càlcul de la normal d’una polilínia qualsevol ..................... 56
20 Formació d’una extrusió .................................................... 57
21 Canvi de direcció ............................................................. 61
22 Escalat ........................................................................... 62
23 Forat ......................................................................... 66
24 Generació d’artifacts al fer forats ................................. 68
25 Generació de formes irregulars al fer forats ................. 69
26 Projecció de la polilínia forat ........................................... 70
27 Tipus d’espeleotemes ....................................................... 73
28 Selecció de polilínia inicial d’un estalactita ..................... 76
29 Selecció del vèrtex simètric a una polilínia ..................... 77
30 Pilar massa inclinat ......................................................... 78
31 Culling .................................................................. 80
32 Triangulació d’una extrusió ......................................... 81
33 Problema al triangular forats ......................................... 83
34 Diferents normals per al mateix vèrtex ......................... 88
35 Ocupació de les AABB .................................................... 89
36 Subdivisió de vèrtexs per aplicar smooth ....................... 93
37 Suavitzat al final de túnel ............................................... 96
38 Textures circulars .......................................................... 97
39 Texturització sense duplicar el primer vèrtex ............... 98
40 Texturització d’una extrusió ......................................... 99
41 Problemes al texturar un forat ...................................... 100
42 Problemes al texturar un forat 2 .................................... 101
43 Texturització del tancament de túnel amb un vèrtex ...... 102
44 Texturització del tancament de túnel amb un vèrtex 2 ..... 102
45 Texturització del tancament de túnel amb n vèrtexs .... 103
46 Texturització del tancament de túnel amb n vèrtexs 2 .... 103
47 Texturització d’espeleotema .......................................... 104
48 Exemple d’espeleotema texturat .................................. 105
49 Comparació de les diferents textures ............................. 106
50 Personatges ................................................................. 108
Índex de taules

<table>
<thead>
<tr>
<th>Núm.</th>
<th>Contenent</th>
<th>Pàgina</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Temps estimat per tasca</td>
<td>25</td>
</tr>
<tr>
<td>2</td>
<td>Temps real per tasca</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>Costos de recursos humans estimats</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>Costos hardware</td>
<td>32</td>
</tr>
<tr>
<td>5</td>
<td>Costos indirectes</td>
<td>33</td>
</tr>
<tr>
<td>6</td>
<td>Costos totals</td>
<td>33</td>
</tr>
<tr>
<td>7</td>
<td>Costos de recursos humans reals</td>
<td>35</td>
</tr>
<tr>
<td>8</td>
<td>Matriu de sostenibilitat</td>
<td>39</td>
</tr>
<tr>
<td>9</td>
<td>Desglòs de la dedicació en hores per tasques i conceptes</td>
<td>123</td>
</tr>
<tr>
<td>10</td>
<td>Desglòs del pressupost per tasques i conceptes</td>
<td>124</td>
</tr>
</tbody>
</table>
1 Introducció i formulació del problema

Actualment la indústria dels videojocs està en una evolució constant, pel que un munt de videojocs són creats contínuament.

Com a tots els desenvolupaments d’un producte, calen persones amb diferents rols. Un dels rols fonamentals en la creació de videojocs és el de generar l’escena que formarà part del món d’aquest. Aquesta tasca ha estat feta manualment per dissenyadors, que s’encarreguen de decidir com volen que sigui el món, així com modeladors, que s’han encarregat de plasmar aquestes idees i crear models per tal de què es puguin utilitzar al videojocs.

D’altra banda, al llarg dels anys s’han anat desenvolupant diferents algoritmes i tècniques per tal de què aquesta generació del món es pugui fer de forma automàtica. Aquestes tècniques són generalment anomenades com a generacions procedurals. D’aquesta forma, fent ús de les generacions procedurals es guanya agilitat en el procés de creació i s’introdueix un factor d’aleatorietat que permet diferenciar entre diversos escenaris ja creats i explorats, evitant així la monotonia i repetició d’aquests i la memorització per part del jugador. D’aquesta forma s’aconsegueix més rejugabilitat en un videojoc.

Per tant, en aquest treball es pretén implementar un algoritme per poder dur a terme una generació procedural de coves, cavernes, mines o masmorres tridimensionals, on el jugador podrà explorar-les per dins.
2 Context

Molts dels videojocs que s’han creat utilitzen models de coves o escenaris semblants com a part del món on es desenvolupa. Tot i així, la majoria d’ells utilitzen models prèviament creats, pel que son molt personalitzables i per tant més flexibles per tenir la forma desitjable. És per això que al utilitzar algoritmes procedurals per la seva creació no s’obtenen formes tant modelables ni amb l’opció de posar tants detalls.

Figura 1: Exemples de diferents videojocs on apareixen coves. Es pot apreciar els diferents detalls a cadascuns difícils d’aplicar a un algorisme procedural
Els algoritmes procedurals[1] van començar a ser utilitzats en l’àrea de gràfics per computador per generar sobretot textures, models o terrenys de forma automàtica i pseudoalatoriament. Per a les textures generalment s’utilitzen fractals o funcions de soroll.

Més tard, van aparèixer a l’àrea dels videojocs com a generació del món on es desenvoluparia l’acció d’aquest, sent anomenats els primers jocs d’aquest estil com a *Roguelike*, on es generaven masmorres on el jugador podia explorar i interactuar amb elements d’aquesta. Al llarg dels anys molts videojocs han estat utilitzant tècniques d’aquest estil[2], per tal de facilitar la creació d’escenaris i intentar evitar ahir un efecte de repetició d’aquests.

En l’àmbit dels videojocs no només s’han utilitzat la generació procedural per a la creació d’escenaris, sinó que també per a altres elements com per exemple la generació de textures, d’enemics o fins i tot per a elements de l’entorn o per a l’àudio.

Per exemple, existeix el cas del polèmic i recent videojoc *No Man’s Sky*, el qual tot el seu contingut està generat de forma procedural, donant així pas a un increïble número de escenaris diferents creats per poder explorar.

Tot i així, en la majoria de casos existents i a diferència del que es vol fer en aquest treball, aquesta generació del món s’ha fet a partir de models tridimensionals o elements creats manualment prèviament i per tant ja existents.

D’aquesta forma, la generació ha consistit en molts casos en una correcta combinació d’aquests models.

### 2.1 Actors implicats

A continuació es detallen totes les persones que puguin estiguin implicades o interessades en aquest projecte.

#### 2.1.1 Dissenyador, Programador, Tester

En aquest cas, la mateixa persona farà el paper de dissenyar, programar i fer proves amb el projecte, degut a què sóc l’únic que el duu a terme. Es podrien ajuntar tots aquests rols com a un únic rol de desenvolupador.
2.1.2 Director del projecte

El director s’encarregarà de supervisar el projecte, assegurant-se així que es compleixi amb el calendari i objectius acordats amb el desenvolupador.

D’altra banda, també s’encarregarà de proposar millores i idees que puguin millorar la qualitat final del projecte.

2.1.3 Desenvolupador extern com a usuari

El resultat final podrà ser utilitzat com una eina per a què altres desenvolupadors de videojocs la puguin integrar, i fins i tot millor-la, en el seu projecte.

2.1.4 Usuari final

Està pensat en ser el jugador que pugui utilitzar tant el projecte actual o projectes derivats.
3 Estat de l’art

Els algoritmes procedurals als videojocs s’han utilitzat per generar sobretot escenaris[3]. Se sol afegir aleatorietat, per tant la majoria de tècniques fan servir funcions de soroll o bé pseudoaleatories[4].

Tot i així, no es poden generalitzar totes en un mateix grup, donades les característiques i els objectius de cadascuna, però també és possible la combinació entre elles.

3.1 2D

Inicialment, la generació només es feia per entorns 2D, degut a les limitacions de l’època. Aquesta generació consistia bàsicament en la generació d’habitacions i en els passadissos que les connectaven[5, 6], utilitzant la triangulació de Delaunay[7]. Per evitar que no s’uneixin totes les habitacions entre elles, es fa servir un Minimum spanning tree [8].

Degut a què l’objectiu principal del projecte és aconseguir un escenari tridimensional, aquests mètodes no es faran servir directament, però pot ser d’ajut tenir-los en compte.

3.2 Unió de mòduls

Aquest mètode fa servir models tridimensionals prèviament creats, anomenats mòduls. D’aquesta forma, l’algoritme s’encarrega de unir-los de forma que es connecten els models que representen habitacions mitjançant models que representen passadissos[9].

Aquesta tècnica era utilitzada sobretot per generar laberints o masmorres i és una evolució directa del mètode explicat anteriorment.

3.3 Heightmap

Per representar terrenys en els videojocs, s’utilitzen els heightmap[10], que venen a ser una matriu quadrada que conté el mapa del terreny, amb un valor a cada posició que representa el valor l’elevació del terreny en aquella coordinada.
Figura 2: Exemple del conjunt de mòduls utilitzats per a un cas del mètode de generació procedural d’unió de mòduls.

Al ser una matriu quadrada, es poden interpretar com a textures, de forma que a més dels propis algoritmes de generació de heightmaps, molts algoritmes de generació procedural de textures poden ser aplicables per generar-los, amb algunes modificacions per a què visualment semblin terrenys.

Degut a què les coves es poden interpretar com a espais tancats, és a dir interiors, aquests mètodes de generació no es poden utilitzar directament, ja que els terrenys són espais exteriors. Tot i així, és un tema molt estudiat actualment en l’àrea de gràfics per computador aplicat als videojocs\[11, 12\].

3.4 L-system

Aquest mètode de generació consta en una sèrie de regles a aplicar a cada punt de la cova, com per exemple, canviar la direcció en la que s’està generant, fer una bifurcació,... D’aquesta forma, es segueix una regla de gramàtiques que va generant cada tros [13, 14].

Tot i haver estat utilitzat en alguns estudis per la generació de coves, aquest mètode és molt freqüent per a la generació d’arbres i plantes.

3.5 Voxels i octrees

Per molts algoritmes procedurals, és molt habitual l’ús de voxels[15], així com de l’estructura de dades octree[16]. Bàsicament, venen a representar una subdi- visió de l’espai tridimensional.
Existeixen diversos estudis previs on s’ha fet l’ús d’aquestes tècniques per la generació de coves[17], alguns fins i tot generant també objectes addicionals dins d’aquestes, com per exemple estalagmites[18].

3.6 Conclusions

El mètode que es vol utilitzar és, a partir d’una entrada de la cova definida simplement per uns punts que formen una polilínia, anar generant les diferents parts d’aquesta de forma que es vagi extrudint el conjunt de punts o polilínia actual. Cada cop que s’extrudeixi, s’aplicaran diferents operacions per tal de què no sigui un simple tub.

Per tant, de les tècniques esmentades la que més s’assembla al que es vol fer és el L-system. L’única similitud real, però, és que en cada moment s’aplicaran diferent operacions per generar una part de la cova d’una forma o d’una altre, però utilitzant funcions pseudoaleatories guiades per regles i restriccions definides. Per tant, no és exactament una tècnica de L-system.

També es podran aplicar algunes idees, així com molts dels mètodes utilitzen octrees per tal de generar la cova, mentre que a aquest projecte no s’utilitzarà per a la generació, sinó que es valorarà l’utilitzar-los per poder marcar una zona de l’espai ja visitada. ajudant
Figura 4: Exemples de dos resultats de generació diferent al utilitzar un mètode que fa servir tant tècniques de L-system com de voxelització.

així a detectar zones on ja hi ha algun tros de la cova generat i evitar les interseccions.

A més, molts dels mètodes que utilitzen aquestes tècniques tenen algunes restriccions, com per exemple tenir un espai limitat sobre el qual es generarà la cova, o un temps de càlcul major degut a la creació dels octrees.

Per tant, cal esmentar que el mètode que es vol seguir no ha estat analitzat abans de la mateixa forma que es vol implementar, pel que es pot considerar una tècnica relativament nova.
4 Objectius del projecte

Un cop introduït aquest treball, s’exposen els diferents objectius inicials que es volien assolir, de forma cronològica:

- Estudiar tant els algoritmes com estructures de dades ja existents utilitzats per generacions procedurals, especialment per coves.

- Investigar la forma natural de creació d’una cova.

- Adaptar la informació obtinguda mitjançant la recerca per tal d’aconseguir generar un escenari semblant a una cova, caverna, mina o masmorra, prenent la exploració d’aquesta com a objectiu principal. L’important serà com es veu des de dins, ja que és per on es suposa que el jugador l’explorarà.

- Per la generació, implementar els algoritmes i estructures de dades el més eficient possibles, tant per generar la cova com per a decidir quina operació de generació aplicar.

- Altres objectius secundaris per tal de millorar la visualització i permetre l’exploració:
  - Obtenir diferents textures a aplicar a la cova per tal d’obtenir un bon aspecte visual.
  - Investigar diferents formes d’aplicar textures a la cova, és a dir, de texturitzar-la.
  - Aplicar una correcta il·luminació a les diferents zones de la cova.
  - Adapar el moviment d’un personatge per poder explorar la cova, així com una càmera que el segueixi correctament.
  - Crear aquest projecte com a una eina de generació de coves de forma que pugui ser utilitzada tant per un usuari final com per a un usuari desenvolupador.

D’aquesta forma, es podria considerar com a objectius principals del projecte el de la generació de la cova, sense tenir en compte la seva visualització, el qual seria un dels principals objectius secundaris.
5 Abast

Abans de començar a desenvolupar el projecte, cal pensar en el flux i passos principals de l’algoritme general, així com definir la metodologia i eines que es faran servir per tal de facilitar i agilitzar tot el procés de desenvolupament.

Com s’ha comentat anteriorment, la generació procedural es farà partint de zero, és a dir, sense utilitzar cap altre model ja existent. Per tant, a partir d’uns punts d’entrada a la cova definits per l’usuari, s’anirà generant la cova de diverses formes possibles, mantenint les estructures de dades necessàries per tal de reduir al màxim el temps de computació total. Aquests punts d’entrada els anomenarem polilínia, ja que formen aquesta figura geomètrica.

Aquesta polilínia s’extrudirà formant una nova polilínia, de forma que a aquesta extrusió se li podran aplicar diverses operacions. Per tant, existiran així diverses formes de generar aquell tros de la cova, amb diferents paràmetres que permetran que l’extrusió sigui d’una forma o d’una altra. Es repetirà diversos cops l’operació d’extrusió fins finalment generar tota la cova. Caldrà, per tant, com a mínim un paràmetre que permeti controlar quants cops s’extrudirà la cova per generar un nou tros d’aquesta.

Durant la generació, s’aniran aplicant les diverses operacions de forma recursiva, decidint a cada pas quina operació o combinacions d’aquestes aplicar. Es provaran diferents formes per tal de decidir quina operació aplicar, des de una simple generació d’un número aleatori fins a tenir en compte les diferents operacions aplicades fins al moment.

També caldrà tenir un mètode per tal d’evitar la intersecció o col·lisió entre diferents zones de la cova.

Degut a què es tracta d’un algoritme que depèn de l’atzar, caldrà fer diverses proves i execucions per verificar que el resultat és el desitjable. Caldrà intentar que s’assemblin mínimament a l’estructura d’una cova, pel que caldrà estudiar la generació d’aquestes i per tant aproximir l’algoritme per tal de què tendeixi a generar la cova d’una forma més o menys general. Tot i així, l’important és generar un espai que doni a impressió d’estar a dins d’una caverna, mina, cova o masmorra, per tant, no es valorarà quin aspecte té vist des de fora, només des
de dins.

També s’estudiarà la possibilitat d’afegir millores visuals a la cova, com per exemple aplicar textures o il·luminació, per tal de apropar el seu aspecte al d’una cova, mina, caverna o masmorra, i donar-li així més detall i realisme.

Tots aquests aspectes estan detallats en els corresponents apartats d’implementació, tenint en compte les possibles alternatives estudiades, així com la justificació de les utilitzades i implementades finalment.

Finalment, en el cas de què s’hagi obtingut uns resultats altament acceptables, es publicarà aquest projecte a la Unity Asset Store.

5.1 Possibles obstacles

A continuació es detallaran possibles obstacles que puguin aparèixer durant el desenvolupament del projecte, així com possibles solucions per poder ser resolts en el mínim temps possible.

5.1.1 Errors al codi

Com a tots els desenvolupaments de software, els petits errors de codi tenen un gran impacte en el producte final. Caldrà anar provant doncs totes les diferents versions del projecte a mesura que es vagi desenvolupant.

5.1.2 Durada del projecte

Serà d’uns tres mesos i mig, per tant es pot considerar el desenvolupament com a un procés que ha de ser necessàriament el més ràpid possible. Les continues reunions amb el director del projecte permetran reforçar aquest aspecte.

5.1.3 Pèrdua del projecte

Degut a la possibilitat del disc dur de ser espatllat o qualsevol altre problema d’aquesta índole, és imperatiu tenir el projecte guardat en algun repositori, o bé algun sistema que permeti fer continues còpies de seguretat.
5.1.4 Artifacts

Al estar generant triangles i altres figures geomètriques, caldrà evitar la creació de possibles Artifacts, ja que poden produir un mal aspecte a la cova.
6 Metodologia i rigor

6.1 Metodologia

Degut al poc temps per desenvolupar el projecte, s’ha optat per seguir una metodologia àgil. Així doncs, cal triar quina d’aquest grup es seguirà segons les circumstàncies donades, descrites a continuació.

6.1.1 Mida de l’equip

Serà un projecte individual, i per tant, desenvolupat per una sola persona en comptes de per un equip sencer. Això sí, comptarà amb el seguiment del director del projecte.

6.1.2 Cicles de desenvolupament

Els cicles de desenvolupament seran curts, amb reunions i sprints cada dos setmanes per veure l’estat del projecte. D’aquesta forma, es proposaran diferents objectius a assolir a cada cicle o reunió.

Amb aquestes reunions continuades es vol aconseguir més flexibilitat a l’hora de fer possibles canvis.

6.1.3 Feedback

Degut a què no hi ha clients directes ni cap persona amb l’únic rol de tester, s’aprofitaran les reunions amb el director del projecte per tenir un feedback del projecte en cada reunió, proposant així possibles canvis i millores a aplicar.

6.1.4 Metodologia final

Així doncs, es farà servir una metodologia Scrum[19] adaptada a un desenvolupament individual, degut a què és la metodologia que més s’adapta a les necessitats i estat del projecte.

Per tal de validar, es comprovarà a cada reunió si s’ha completat les tasques planificades per al sprint actual. Tot i intentar evitar-ho, si alguna tasca no ha pogut ser finalitzada en el termini acordat, s’establirà per al següent sprint.
6.2 Eines de seguiment i validació

6.2.1 Control de sprints

Per tal de controlar les tasques a fer a cada sprint, s’utilitzarà la pàgina web http://www.scrumme.com.br, que permet guardar en línia i de forma compartida les tasques a fer i el seu estat, així com el temps de dedicació en hores esperat.

Per tant, a cada una de les reunions bimensuals es revisarà el compliment de les tasques establertes anteriorment i s’establiran les tasques del següent sprint utilitzant aquesta eina.

6.2.2 Control de versions

S’utilitzarà Git com a eina de control de versions, així com un repositori ubicat a GitHub.

D’aquesta forma, i juntament amb les reunions amb el director del projecte, es tindrà un control de com ha anat evolucionant el project durant el seu desenvolupament.

6.3 Eines i llenguatge de programació

Per tal de facilitar el desenvolupament del projecte es van pensar inicialment dos eines:


- **Blender**[21] És una eina per crear models tridimensionals i animar-los, entre altres coses. Es pot utilitzar amb el llenguatge de programació Python

Finalment, es va decidir utilitzar Unity i el llenguatge C#, degut a què un cop finalitzat tot el projecte, es pot adaptar i utilitzar fàcilment a qualsevol altre projecte fet amb Unity. A més a més, facilita i agilitza molta feina de baix nivell, ajudant així a poder enfocar-nos més en els objectius propis del projecte. El fet de triar aquest llenguatge és perquè és el més usat, pel que hi ha més documentació i solució de problemes disponible a la xarxa.
Figura 5: Percentatge d’úsabilitat dels diferents llenguatges de Unity a finals del 2014, on Boo encara existia

6.4 Desviacions

Tant la metodologia com les eines triades han permès la flexibilitat en alguns canvis fets a la planificació, així com optimitzar els diferents aspectes de la implementació del projecte, pel que no s’ha vist necessari el canviar-les.

Ha ajudat molt les constants reuniions cada dues setmanes per tal de poder ensenyar al director del projecte els canvis i les diverses implementacions, ja que s’ha obtingut un bon feedback. Especialment ha anat bé per tal de poder ajudar a la pressa d’algunes decisions, no només d’implementació sinó també de planificació. La freqüència d’aquestes reuniions s’han mantingut com s’havien planificat inicialment.

A mesura que s’han anat fent les reuniions, s’ha guardat el registre d’aquestes mitjançant el projecte de la plana web scrumme, així com diverses opcions i decisions d’implementació i les hores dedicades a cada sprint. Va existir el risc i el pànic que durant un temps els servidors d’aquesta plana no funcionaven i per tant no es podia accedir a la informació guardada. Aquest contratempo es va acabar resolent sense cap repercussió final.

Finalment, per a tal de fer l’edició dels dibuixos explicatius per donar suport a les explicacions, s’ha fet servir el software gratuït Gimp, el qual no afegix cap tipus de cost.
7  Planificació temporal

7.1  Duració

La durada del projecte és d’uns quatre mesos aproximadament, del 13 de febrer fins al 26 de juny del 2017. Degut a què és de 18 crèdits ECTS, la duració ha d’oscil·lar entre 450 i 540 hores totals, acceptant possibles variacions sempre i quan s’assoleixen els objectius del projecte.

7.2  Pla de projecte

Degut a utilitzar una metodologia àgil **Scrum** amb objectius a complir cada dues setmanes, no es definirà un pla estàtic ni tasques massa concretes. Per tant, cada dues setmanes s’anirà definit amb detall el següent sprint amb els objectius específics, així com anar proposant possibles millors al treball ja realitzat. La data de la primera d’aquestes reunions serà el 20 de febrer.

Està planificat en dedicar-hi entre 25 i 40 hores setmanals al projecte, dependent del temps necessari per a cada sprint. D’aquesta forma, es permet la flexibilitat i l’existència d’alguna desviació temporal de les tasques del pla proposat a continuació.

7.2.1  Preparació de l’entorn

Primer de tot, caldrà preparar tot l’entorn de treball pel projecte segons els recursos necessaris\(^1\).

Aquesta part també té en compte la pressa de contacte amb el programa *Unity*, que serà el que es farà servir principalment per desenvolupar tot el projecte.

7.2.2  Fita inicial

Aquesta part tracta de planificar el projecte, així com establir els objectius principals, entre altres aspectes d’aquesta índole.

És la part corresponent al GEP, i té les següents divisions:

\(^1\)Esmentats a l’apartat 7.5
• Abast del projecte, així com la contextualització i estat de l’art d’aquest
• Planificació temporal
• Gestió econòmica i sostenibilitat

7.2.3 Anàlisi i Implementació

Inicialment, es farà un estudi i un plantejament de com serà l’algoritme general, és a dir, decidir de quina forma es generarà la cova sense entrar en detalls massa concrets.
Després, a mesura que es vagin fent les diferents reuniions, s’anirà realitzant un anàlisi general del següent pas a implementar, de forma que un cop finalitzades les reuniions s’acabi de perfilar aquest anàlisi i disseny per finalment implementar-lo i fer proves de forma paral·lela. Amb aquest mètode de treball es podrà anant comprovant que hi ha un resultat esperat a mesura que es vagi avançant en el projecte.
D’aquesta forma s’espera idealment seguir la distribució de temps aproximada segons la teoria de projectes: un 40% en el disseny i possibles modificacions, un 20% en la implementació i el 40% restant en les proves.

Es poden distingir dos tipus diferents d’implementacions:

• Primer, es treballarà la generació de la cova com a tal, és a dir, degut a què és una tasca gràfica, caldrà generar el conjunt de triangles que representin el model de la cova de forma correcte, amb un resultat que permeti donar al jugador la sensació d’estar dins d’una cova, caverna, mina o masmorra.
Per tal de dur a terme aquesta generació, caldrà tenir en compte dos grups de funcions:
  – Les funcions de generació, s’encarregaran de generar un tros de la cova d’una forma o una altra, segons l’operació a aplicar a l’extrusió o tros actual.
  – Les funcions de pseudoaleatorietat, degut a què es tracta d’una generació procedural. Aquestes seran les que s’encarreguin de decidir a cada moment quina operació de generació s’aplicarà.

• La segona implementació serà considerada com una extra ja que serà de cara a tenir una millor visualització: aplicar de forma correcta textures
a la cova, així com il·luminació. També s’estudiarà el poder moure un personatge per dins d’aquesta.

Per tant, existeixen certes dependències: no es començarà a implementar funcionalitats extres fins que el generador no estigui acabat, no es podrà començar a implementar fins que l’entorn no estigui configurat,... Tot i així, segons la planificació de Gantt no hi ha grans dependències, ja que s’ha planificat com tasques bastant seqüencials.

7.2.4 Fita final

Aquesta part es duu a terme quan finalment ja s’ha acabat el projecte, i s’està acabant de redactar la memòria. La memòria final contindrà totes les parts del algoritme ben detallats, amb els respectius costos computacionals.

Per tal de fer un correcte seguiment, la redacció de la memòria s’anirà fent en paral·lel a les altres tasques.

A més, un cop acabat d’implementar el generador de coves, està pensat posar-lo a la Unity Asset Store. Per aquesta raó, serà inclòs un annex amb una petita descripció de la usabilitat del programa per a l’usuari, així com una explicació dels diferents paràmetres necessaris que intervenen en la generació.

7.3 Temps de dedicació aproximat

<table>
<thead>
<tr>
<th>Tasca</th>
<th>Temps (h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fita inicial</td>
<td>70</td>
</tr>
<tr>
<td>Preparació de l’entorn</td>
<td>10</td>
</tr>
<tr>
<td>Anàlisi i Implementació Generador</td>
<td>350</td>
</tr>
<tr>
<td>Anàlisi i Implementació Extres</td>
<td>90</td>
</tr>
<tr>
<td>Fita final</td>
<td>50</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>570</strong></td>
</tr>
</tbody>
</table>

Taula 1: Taula amb el temps estimat per a cada tasca. El desglòs es troba a l’apèndix A
7.4 Valoració d’alternatives

Com s’ha comentat, degut a utilitzar la metodologia Scrum, poden haver-hi alguns petits canvis sobre la planificació original. Gràcies a les reunions i sprints que es faran cada dos setmanes, es podrà solucionar ràpidament aquests possibles canvis i evitar així els possibles contratemps.

D’altra banda i com a conseqüència també de la metodologia escollida, el temps de dedicació a cada tasca pot variar, podent fins i tot tenir la necessitat de realitzar més d’una tasca en paral·lel. Les dues tasques que poden tendir més a solapar-se són diferents tasques d’implementació entre elles, o bé una tasca d’anàlisi amb una d’implementació. Això és degut a què pot ser que, segons les proves fetes amb un prototipus, es valori que hi pugui existir una solució millor.

Degut a què la implementació de funcionalitats extres no són tan essencials com la generació de la cova, en cas de necessitar més temps per a la generació, s’utilitzaran hores planificades per a les funcionalitats extres.

Tot i així, per tal de no trobar el problema de la falta de temps de cara al final del projecte, per a cada sprint es valorarà si calen hores addicionals a les planificades. En el cas de què hi hagi hagut una sobreestimació del temps necessari, es començaran les tasques planificades per al següent sprint.

7.5 Recursos

7.5.1 Humans

Com s’ha comentat, tots els rols implicats en el desenvolupament seran fets per una mateixa persona. Tot i així, les tasques que correspondrien als diferents rols segons el diagrama de Gantt serien:

- **Cap de projecte** Correspon a la supervisió de totes les tasques, pel que està implicat tant directe com indirectament a totes les relacionades amb el projecte.

- **Dissenyador** Correspon a totes les tasques referides al Gantt com anàlisi,
per tant s’encarrega del disseny d’algoritmes i d’estructures de dades.

- **Programador** Relacionat amb les tasques d’implementació.

- **Tester** Participa a totes les tasques marcades com a Proves, ja que s’encarrega de comprovar que els resultats són els que s’esperen, així com la localització de possibles bugs.

- **Client** Està relacionat amb les tasques del Tester, degut que es d’on s’obté el feedback del projecte.

Tot i així, el rol que vindria a ser el cap de projecte i client es compartirà en certa manera amb el director del projecte, ja que a cada reunió s’ensenyarà el treball realitzat, així com es decidirà quines tasques seran necessàries per al següent sprint.

### 7.5.2 Hardware

S’utilitzarà un ordinador portàtil per realitzar totes les tasques requerides, tant per la realització del generador de coves com per a la memòria.

### 7.5.3 Software

S’utilitzaran les següents eines software:

- Sistema operatiu *Windows 8.1*

- Motor de videojocs *Unity*, amb la versió 5.5.2. Permet agilitzar i facilitar totes les tasques gràfiques.

- *Monodevelop*, com a editor de codi.

- *Git*, com a controlador de versions de codi

- *L\*T\*P\*X*, per a la edició de la memòria
7.6 Diagrama de Gantt

Figura 6: Diagrama de Gantt
7.7 Desviacions i conclusions

Durant el desenvolupament del projecte s’ha intentat seguir la planificació inicial. Tot i així, hi ha hagut una sobreestimació de la finalització de tasques. És a dir, seguint el diagrama de Gantt, es planificava acabar amb la generació de la cova a mitjans de maig, i començar en aquell moment les implementacions extres. Al haver fet una generació de la cova mínimament acceptable a inicis d’abril, es va decidir començar amb les implementacions extres de millores visuals mentre es millorava el resultat de la generació paral·lelament. Això ha permès poder afegir altres extres que inicialment no estaven planificats com la creació d’estalagmites, estalactites i pilars, així com la possibilitat d’afegir altres objectes de decoració a la cova o bé poder aplicar noves operacions de cara a una hipotètica ampliació i vida futura del projecte.

Algunes de les millores de la generació han sigut aplicar un suavitzat a la cova per tal de què no quedés tan poligonal.

Figura 7: Graf de dependències que s’ha seguit per tal de fer les tasques d’implementació. Es pot observar que hi ha molt paral·lelisme a diferència del que s’havia estimat. La línia discontinua indica la revisió d’apartats anteriors, ja que en comptes d’esperar-se a acabar una tasca completament, el que s’ha fet és tenir una primera versió incompleta i s’ha anat millorant

Una altra millora no relacionada directament amb cap apartat sinó amb tot
el projecte en general, ha sigut el fer ús de corutines per tal de ensenyar pas a pas com es va fent la generació de la cova.

Tot i així, totes aquestes desviacions no han suposat cap variació excessiva en els costos del projecte, ja que els recursos tan materials com humans han continuat sent els mateixos.
Pel que fa als objectius, s’han afegit els objectius extres prèviament esmentats.

A l’entrega del document de seguiment, ens trobàvem en la última etapa de ambdues generacions: resoldre petits bugs i acabar de polir alguns detalls, així com de la redacció de la memòria final. Encara faltava fer l’estudi de si finalment posar el projecte a la *Unity Asset Store*, així de com s’hauria de fer. Aquesta última tasca, però, ja estava planificada per fer-la a mitjans de juny, pel que no suposa cap problema de moment.
A més, degut a què es van assolir perfectament els objectius principals i alguns dels extres, s’ha decidit no afegir més extensions al projecte a partir d’uns dies previs a la fi de seguiment, és a dir, a mitjans-finals de maig. D’aquesta forma, un cop entregada la fi de seguiment, només queda finalitzar la memòria i preparar la presentació.

Finalment, s’ha decidit no penjar el projecte a la *Unity Asset Store* durant la última etapa del projecte ja que s’ha volgut centrar en la finalització de la memòria final, així com de la preparació de la presentació. S’ha considerat, però, penjar aquest projecte a la tenda un cop acabada la data de lliurament com a un projecte personal.

Com es pot observar les hores finals de dedicació han estat de 390, menys que el límit d’hores necessaris del que s’havia estimat. Això ha estat bàsicament degut a la sobreestimació del temps de dedicació al estar pensat aquest com a 8 hores diàries aproximadament. A més, amb el temps dedicat hi ha hagut el compliment dels objectius que s’havien planificat i fins i tot l’afegiment alguns extres, pel que es pot concloure com un fet altament positiu.

Tot i així han hagut dues tasques on s’ha subestimat el temps de dedicació. La primera d’elles es tracta a la preparació de l’entorn que, degut a diversos
errors del hardware especialment pantalles blava\textsuperscript{2}, s’ha hagut de dedicar temps extra a esperes, reinstal·lació i formateig de l’equip utilitzat.
L’altre tasca es tracta de la fita final, a la qual s’han hagut de dedicar més hores de les previstes per al correcte acabament de la memòria, així com de la preparació de la presentació.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|}
\hline
Tasca & Temps (h) \\
\hline
Fita inicial & 58 \\
Preparació de l’entorn & 23 \\
Anàlisi i Implementació Generador & 172 \\
Anàlisi i Implementació Extres & 57 \\
Fita final & 80 \\
\hline
Total & 390 \\
\hline
\end{tabular}
\caption{Taula amb el temps real per a cada tasca. El desglòs es troba a l’apèndix A}
\end{table}

\textsuperscript{2}Error molt comú en el sistema operatiu Windows, que impedeix l’inici d’aquest
8 Gestió econòmica

Per tal de calcular els pressupost total del projecte, es tindran en compte tots els recursos esmentats\(^3\), obtenint així una estimació final.

Els costos es dividiran segons diferents aparts, tenint inclòs a cadascun l’IVA i altres impostos.

8.1 Costos de recursos humans

Tot i fer els diferents rols del desenvolupament de projecte una mateixa persona, els costos seran dividits per rol. Les hores estan calculades a partir del diagrama de Gantt, així com de la distribució de les tasques d’aquest segons el rol\(^4\):

<table>
<thead>
<tr>
<th>Rol</th>
<th>Preu(€/h)</th>
<th>Temps total(h)</th>
<th>Cost total(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap de projecte</td>
<td>80</td>
<td>190.5</td>
<td>15,240</td>
</tr>
<tr>
<td>Dissenyador</td>
<td>45</td>
<td>102.5</td>
<td>4,613</td>
</tr>
<tr>
<td>Programador</td>
<td>40</td>
<td>112.5</td>
<td>4,500</td>
</tr>
<tr>
<td>Tester</td>
<td>25</td>
<td>164.5</td>
<td>4,113</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>570</td>
<td>28,465</td>
</tr>
</tbody>
</table>

Taula 3: Taula amb els costos de recursos humans estimats. El desglòs es troba a l’apèndix B

8.2 Costos de hardware

Com s’ha comentat en apartats anteriors, es farà ús d’un sol ordinador per la realització de totes les tasques necessàries. Es tracta d’un portàtil ASUS A550L.

<table>
<thead>
<tr>
<th>Producte</th>
<th>Preu(€)</th>
<th>Unitats</th>
<th>Vida útil</th>
<th>Amortització(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>799</td>
<td>1</td>
<td>5 anys</td>
<td>43.79</td>
</tr>
<tr>
<td>TOTAL</td>
<td>799</td>
<td></td>
<td></td>
<td>43.79</td>
</tr>
</tbody>
</table>

Taula 4: Taula amb els costos de hardware. El desglòs es troba a l’apèndix B

\(^3\)Corresponents al apartat 7.5

\(^4\)Corresponent a l’apartat 7.6 i 7.5.1 respectivament

\(^5\)Obtingut a partir de la informació de l’empresa Iteral

32
8.3 Costos de software i llicències

En quant al software utilitzat, el cost total és 0 ja que és tot gratuït. Pel que fa a les llicències, són totes gratuïtes excepte la de Windows 8. Tot i així, ja venia inclosa amb la compra del PC, per tant ignorarem el seu cost. Així doncs, es pot considerar com a nul tot el cost del software.

8.4 Costos indirectes

Cal tenir en compte algunes altres despeses generals que estan presents a qualsevol projecte actual, segons la durada del projecte:

<table>
<thead>
<tr>
<th>Producte</th>
<th>Preu (€) (^7)</th>
<th>Unitats</th>
<th>Cost aproximat (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electricitat</td>
<td>0.1174/kWh</td>
<td>55 kWh</td>
<td>6.45</td>
</tr>
<tr>
<td>Internet</td>
<td>45/mes</td>
<td>4 mesos</td>
<td>180</td>
</tr>
<tr>
<td>Transport</td>
<td>4/viatge</td>
<td>11</td>
<td>44</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td>230.45</td>
</tr>
</tbody>
</table>

Taula 5: Taula amb els costos indirectes. El desglòs del transport es troba a l’apèndix B

8.5 Pressupost total

Finalment, es té el següent pressupost total tenint en compte tot el necessari per al projecte:

<table>
<thead>
<tr>
<th>Tipus de cost</th>
<th>Cost aproximat (€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Recursos humans</td>
<td>28.465</td>
</tr>
<tr>
<td>Hardware</td>
<td>43.79</td>
</tr>
<tr>
<td>Software</td>
<td>0</td>
</tr>
<tr>
<td>Indirectes</td>
<td>230.45</td>
</tr>
<tr>
<td>TOTAL</td>
<td>28.739.24</td>
</tr>
</tbody>
</table>

Taula 6: Taula amb els costos totals

\(^6\)Obtingut a partir de la informació del producte

\(^7\)Obtingut a partir de la informació de les factures
Com es pot observar, es un preu econòmicament competitiu. Una de les raons d’aquest preu és el utilitzar software gratuït i pocs recursos degut a la mida de l’equip de projecte.

8.6 Control de gestió

De la mateixa forma que a la planificació, el principal problema pot venir de la desviació temporal d’alguna tasca, és a dir, de la necessitar de ampliar les hores per alguna des les tasques. Tot i així, s’utilitzarà la planificació del diagrama de Gantt i la metodologia seguida per intentar corregir aquestes possibles desviacions sense que es tingui un gran impacte en el cost final.

Per tal de tenir un control de les desviacions respecte el pressupost proposat, a mesura que es vagin completant les activitats s’anirà actualitzant el temps real dedicat, obtenint alhora el cost real.

Un cop s’hagin finalitzat totes les tasques, es compararà el pressupost planificat respecte el cost real, fent la diferència. D’aquesta forma es podrà valorar la desviació no només total, sinó també per cada tasca i concepte individual. Finalment es podrà veure si els costos reals han sigut majors o menors del pressupost que s’ha proposat.

8.7 Desviacions

Degut a què la variació del cost econòmic d’aquest projecte depèn essencialment de la dedicació en hores dels recursos humans, com aquestes han sigut menors de les planificades, hi ha hagut un guany econòmic respecte al que s’havia previst.

Tot i així, el rol que resulta més costós és el que més s’ha aproximat a la estimació feta a l’inici, pel que això ha provocat que el guany econòmic no fos tan exagerat.

Es pot concloure que en un projecte real aquest fet seria una gran fita al tenir guanys en comptes de pèrdues, pel que es pot valorar com una cosa molt positiva.
Taula 7: Taula amb els costos de recursos humans reals. El desgàlós es troba a l'apèndix B

<table>
<thead>
<tr>
<th>Rol</th>
<th>Preu(€/h)</th>
<th>Temps total(h)</th>
<th>Cost total(€)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cap de projecte</td>
<td>80</td>
<td>181</td>
<td>14,480</td>
</tr>
<tr>
<td>Dissenyador</td>
<td>45</td>
<td>58</td>
<td>2,610</td>
</tr>
<tr>
<td>Programador</td>
<td>40</td>
<td>71</td>
<td>2,840</td>
</tr>
<tr>
<td>Tester</td>
<td>25</td>
<td>815</td>
<td>2,025</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>390</td>
<td>21,955</td>
</tr>
</tbody>
</table>
9 Sostenibilitat

9.1 Econòmica

A l’apartat 8 s’ha detallat tots els costos, tant de recursos materials com humans, implicats en el projecte. També s’han tingut en compte les possibles desviacions d’aquests, així com ajustos que es podien anar aplicant a mesura que es va desenvolupant.

Es podria realitzar el projecte amb un equip una mica més gran per tal de reduir la seva duració. Això, però, implicaria un augment en els costos indirectes i de hardware proporcional al nombre de persones afegides al projecte, així com als recursos humans. Aquest increment en recursos humans per tant hauria de ser bastant baix per tal de què realment s’aconseguís reduir el cost del projecte. Un altre factor que reforça aquest fet és la complicació de dividir tasques de forma independent segons com s’ha plantejat el projecte.

D’altra banda, el temps dedicat a cada tasca és la correcte, segons la importància d’aquestes de cara al projecte. A més a més, al voler implementar una tècnica de generació procedural relativament nova, serà suficient amb conèixer les idees generals d’altres mètodes de generació. Al ser una tècnica diferent i implementada amb Unity, es pot compartir a través de la Unity Asset Store per tal que qualsevol desenvolupador pugui utilitzar-la per la seva conveniència, tenint així una col·laboració indirecta.

Degut al caire del projecte, els costos econòmics durant la vida útil seran els mateixos que els de la seva producció, ja que una de les seves utilitzacions és la de millorar l’eina implementada. La utilització directe del projecte tal i com existeix sense haver d’aplicar cap millora, permetria un estalvi econòmic en quan a no requerir tanta necessitat d’un modelador, pel que podria aportar als equips de desenvolupament un estalvi en aquest aspecte.

Un risc econòmic, però improbable seria el fet de què el programa Unity quedés en desús, cosa que tot el codi implementat no serviria de gaire, i quedaria
migrar-lo a una altra plataforma. Tot i així, els algoritmes creats es podrien reaprofitar des de el punt de vista teòric. Un altre risc seria el fet de què una generació procedural no té el mateix resultat que una manual, pel que alguns usuaris podrien quedar descontents.

9.2 Social

Actualment, la situació del país és de precarietat laboral i corrupció política, sortint encara d’una crisi econòmica. Respecte al sector dels videojocs, és un sector en constant evolució. Fins feia poc, tenia poc impacte al país, però últimament està sent reforçat en part gràcies a la evolució i importància a nivell global del sector.

La realització d’aquest projecte no té cap impacte negatiu en aspectes ètics. L’estudi de noves tècniques, així com la pràctica i l’aprenentatge adquirits al projecte es podria considerar com una gran millora a nivell personal i professional.

No hi ha una demanda directe de la realització del projecte, però qualsevol tècnica nova i diferent ajuda sempre a impulsar un sector i és interessant l’existència de la varietat, més en el sector del qual estem tractant.

Es podria veure afectat negativament el col·lectiu dels modeladors d’objectes i dissenyadors d’escenaris, però amb un impacte negligible degut a què existeix la necessitat de modelar altres objectes, així com l’existència d’altres tipus d’escenaris.

Existeix la discussió polèmica sobre l’impacte dels videojocs a la societat, existint sobretot estudis que ho defensen com un impacte positiu, especialment en capacitat cognitives. També hi ha col·lectius que ho defensen com un impacte negatiu, especialment en termes de dependència, tot i que cada cop gràcies als estudis anomenats anteriorment, la mida d’aquesta població està descendent.

Tot i així, l’existència d’aquest projecte tampoc té un gran impacte directe
en el sector, però es poden prendre com a referència les discussions i impactes generals del sector per aplicar-les al projecte.

9.3 Ambiental

Els recursos necessaris durant tot el projecte relatiu al medi ambient, només tenen a veure amb l’electricitat i Internet utilitzats per al seu desenvolupament. Per tant, l’única contaminació ambiental prové d’aquest consum. Un cop acabat el projecte, si es vol fer ús dels resultats obtinguts, caldrà al menys un ordinador, per tant caldrà utilitzar l’electricitat de nou com a recurs.

Tot i així, al utilitzar només un ordinador en el projecte, l’estalvi energètic obtingut a nivell global de cara a no realitzar el TFG seria nul. Igualment, l’única forma de realitzar aquesta activitat és mitjançant un ordinador, per tant, la realització d’aquest TFG no suposa en cap sentit un perill mediambiental.

En quant als materials utilitzats, l’ordinador ja estava prèviament comprat, i podrà seguir sent utilitzat un cop acabat el projecte.

Per tant, tot i tenir la necessitat de consumir electricitat quasi no té cap impacte en el medi ambient, degut a ser un projecte purament de software i a la seva escala.

La vida útil d’aquest projecte durarà mentre algun altre desenvolupador expandeixi el projecte o bé l’utilitzi per a generar elements del seu joc, pel que el consum i per tant l’impacte ambiental seguirà sent el mateix. L’ús d’aquest projecte no tindrà cap millor directa mediambiental, i els recursos guanyats per tal de no haver de fer la generació manualment equivalen als que s’utilitzarien per poder millorar el projecte, tot i que a la llarga si s’obtinguessin bons resultats si que podria existir un possible benefici energètic.

L’únic risc ambiental d’aquest projecte es deu al seu consum energètic, però com s’ha justificat anteriorment, es quasi nul.
9.4 Conclusions

A partir dels raonaments i justificacions dels apartats anteriors obtenim la puntuació següent per a la matriu de sostenibilitat:

<table>
<thead>
<tr>
<th>Sostenibilitat</th>
<th>Econòmica</th>
<th>Social</th>
<th>Ambiental</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Projecte en producció</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>26</td>
</tr>
<tr>
<td>Vida útil</td>
<td>18</td>
<td>17</td>
<td>17</td>
<td>52</td>
</tr>
<tr>
<td>Riscs</td>
<td>-4</td>
<td>-2</td>
<td>-1</td>
<td>-7</td>
</tr>
<tr>
<td>Total</td>
<td>23</td>
<td>23</td>
<td>25</td>
<td>71</td>
</tr>
</tbody>
</table>

Taula 8: Matriu de sostenibilitat

Com es pot observar, la puntuació final de sostenibilitat és bastant elevada respecte al possible rang (-60,90), pel que es pot concloure com un projecte altament sostenible, implicant així una gran satisfacció a nivell personal.

El sector dels videojocs està en constant creixement amb diverses opinions sobre el seu ús, però aquest anàlisi demostra en certa manera que se li ha de donar recolzament i que el seu desenvolupament pot aportar diverses millores en molts aspectes.
10 Identificació de lleis i regulacions

Aquest projecte no utilitza dades personals ni de caires semblants, pel que no s’ha de preocupar de complir les lleis relacionades amb aquest aspecte al no tenir-hi cap repercussió.

És un projecte open source, ja que el repositori és públic per a tothom, per tant no cal preocupar-se tampoc per temes de copyright i semblants. Si s’acaba posant a la Unity Asset Store si que s’haurà de mirar aquest tema, ja que deixaria de ser open source al entrar en àmbit comercial.

Finalment, pel que respecta a la utilització de material prèviament creat per una font externa, només s’ha de tenir en compte les textures utilitzades, ja que és l’únic material no creat per el propi software ni per cap integrant del projecte. S’ha assegurat el utilitzar textures open source i per tant gratuïtes a Internet per un ús no comercial. Han estat obtingudes per dos fonts diferents, per una banda per la publicació de filtres i textures creades per la comunitat de Filter Forge[22], un plugin del programa Adobe Photoshop que permet la creació d’aquestes. D’altra banda, també s’ha utilitzat les textures creades per la comunitat CGTextures[23].

Pel cas de què es volgués comercialitzar el software creat utilitzant aquest contingut creat amb una certa normativa de copyright, en un principi s’hauria de deixar d’utilitzar aquestes textures al no tenir els drets d’autors necessaris per la seva comercialització. Una altra alternativa que inclou el possible ús d’aquestes textures al producte, seria estudiar com serien les despeses econòmiques després d’haver contactat amb els autors d’aquestes sobre com retribuir-los o bé poder arribar a una altra mena d’acord per utilitzar material creat per ells.

Així doncs, el projecte no infligeix cap llei de propietat intel·lectual ni normativa, al ser un projecte purament amb elements propis i sense necessitat de dades externes.
11 Justificació del projecte

En els subapartats següents es justificarà la relació del projecte amb els estudis i coneixements adquirits a l’especialitat del grau.

11.1 Assignatures relacionades amb Computació

Les assignatures cursades a l’especialitat de computació que es creuen que seran d’utilitat per al projecte són:

- **Gràfics** La més clara de totes ja que el que es vol fer es tracta d’una tasca gràfica, per tant caldrà tenir un bon coneixement de com funciona tot a nivell intern, especialment en termes d’eficiència. Per exemple, conceptes com el *z-fighting*, les propietats d’un material o la texturització ajudaran també a saber si s’està generant la cova correctament.

- **Algorismia** Per saber avaluar correctament el cost d’un algoritme o estructura de dades, així com saber creuar-los correctament. Aquesta assignatura es centra molt en a partir d’un problema saber trobar una solució que s’adequiés, pel que servirà molt els coneixements adquirits.

- **Llenguatges de programació** Per ajudar a conèixer les característiques del llenguatge en el qual es desenvoluparà el projecte, C#, i facilitar així el desenvolupament amb aquest.

- **Teoria de la computació** Inicialment es pensava que els coneixements adquirits poden ajudar a establir l’ordre en el que es podran aplicar les diferents funcions de generació, mitjançant màquines d’estat o regles. Tot i així, al final s’ha decidit utilitzar un altre mètode per generar les diferents operacions.

11.2 Competències tècniques del projecte

A continuació es llisten les diferents competències tècniques triades per al projecte amb el seu nivell d’assoliment, juntament amb una justificació de perquè s’ha triat cadascuna i com es preven assolir aquest nivell d’assoliment:

- **CCO1.1: Avaluar la complexitat computacional d’un problema, conèixer estratègies algorísmiques que puguin dur a la seva resolució, i recomanar, desenvolupar i implementar la que garanteixi el millor rendiment d’acord amb els requisits establerts.**
Caldrà estudiar i implementar la millor solució en termes d’eficiència per tal de fer el generador el més computacionalment senzill possible (menys costós en temps i memòria) segons els objectius definitis. Serà un punt clau ja que en cada apartat diferent de l’algoritme, així com de forma general, es procurarà aplicar la millor solució.

**• CCO1.3: Definir, avaluar i seleccionar plataformes de desenvolupament i producció hardware i software per al desenvolupament d’aplicacions i serveis informàtics de diversa complexitat.** [Bastant] Caldrà triar la plataforma software de desenvolupament que més avantatges proporcioni per al que es vol fer, així com un hardware que s’adequï i pugui executar correctament aquesta plataforma. La justificació de per què s’ha acabat triant les plataformes escollides i no unes altres es troba a l’apartat de metodologia.

**• CCO2.6: Dissenyar i implementar aplicacions gràfiques, de realitat virtual, de realitat augmentada i videojocs.** [En profunditat]: Degut a què s’ha d’implementar un generador d’un model tridimensional, tot gira entorn a dissenyar i implementar una aplicació gràfica, així com la seva relació amb els videojocs. Per tant, aquesta competència és la que més es treballarà.

**• CCO3.1: Implementar codi crític seguint criteris de temps d’execució, eficiència i seguretat.** [Una mica] El generador només s’executarà un cop en tot el programa, és a dir, no s’està executant contínuament a cada frame. Per això, no es tan crític la seva execució, però si que ha de tenir un temps d’execució i eficiència mínimament acceptables, el que vindria a ser el temps de càrrega de l’escena a un videojocs. Els aspectes de seguretat no tenen cap relació amb aquest projecte.

### 11.3 Adequació a Computació

El projecte consisteix en desenvolupar una aplicació gràfica relacionada amb la programació de videojocs tenint en compte diferents criteris, especialment l’eficiència. Per tal d’agilitzar i facilitar el desenvolupament s’ha escollit una plataforma específica dedicada a la producció de videojocs.

Es proposa una tècnica innovadora de generació de coves 3D procedural que cal estudiar i avaluar correctament per tal d’implementar-la amb uns re-
sultats satisfactoris. Es tenen en compte tant els seus avantatges com els seus límits. D’aquesta forma, s’aconsegueix una solució informàtica a un problema en l’àmbit dels videojocs amb els raonaments i coneixements pròpiament adquisits a l’especialitat de computació.

Finalment, per dur a terme aquest projecte s’utilitzaran els algoritmes i estructures de dades necessaris segons l’anàlisi realitzat, procurant evitar un cost computacional massa alt. La implementació dels algoritmes estarà basada en decisions pseudoaleatories i diverses condicions d’elevada complexitat.

Per totes aquestes raons, aquest projecte està suficientment justificat per a pertànyer a l’especialitat de Computació.
12 Mètode de Generació

Aquest apartat pretén explicar de forma general l'algoritme que s'utilitza per a la generació de la cova, sense entrar en detalls tècnics o d’implementació.

La part fonamental de l’algoritme de generació és la de l’extrusió d’un conjunt de de punts que formen una polilínia per tal de generar-ne una de nova; tot girarà en voltant de repetir aquesta operació. A continuació s’explica com formar des de parts simples de la cova a la cova final, en ordre ascendent de grau de complexitat. En aquest apartat no es tindrà en compte elements decoratius com les textures o les estalagmites, ja que es consideren elements extres i aquest apartat serveix només per donar una breu idea de com funciona la generació.

12.1 Túnel simple

Es podria considerar el cas més bàsic com el de una cova en la qual no es genera cap forat, de forma que s’obtindria el que vindria a ser un túnel. Per fer-ho més simple encara, serà un túnel senzill al qual no se l’hi ha aplicat cap operació, i per tant té una forma semblant a la de un tub.

Figura 8: Exemple d’un túnel simple, amb 100 extrusions. Degut a què no s’està aplicant cap operació, el resultat sempre serà el mateix.

Per generar aquest túnel amb la forma de tub, es partirà d’una polilínia inicial, que es pot considerar com l’entra de del túnel. Aquesta polilínia formarà una de nova mitjançant una extrusió: cada punt es replicarà a una certa distància i en una direcció des de cada punt original. D’aquesta forma es gene-
rarà la següent polilínia amb el mateix número de punts que la seva predecessora.

Figura 9: Exemples de com es van generant cada polilínia a partir de l’extrusió de l’anterior: els punts grocs representen els vèrtexs de cada polilínia, i les línies blaves la triangulació entre ells.

Repetint aquesta operació d’extrusió diversos cops, s’obtindria el túnel simple.

12.2 Túnel complex

En el cas anterior s’ha explicat com es genera un túnel mitjançant extrusions simples: la direcció i distància d’extrusió era constant durant la creació de tot el túnel. Ara bé, a les extrusions se li poden aplicar diferents canvis per tal de tenir un resultat més irregular i no obtenir sempre el mateix tub allargat. Aquests canvis a la forma de les extrusions seran anomenats operacions, amb un nom bastant auto descriptius, i s’esmenten a continuació:

- Escalat
- Rotació
- Canvi de direcció
- Canvi de distància
• Creació d’estalagmites, estalactites i pilars

La implementació i el detall d’aquestes operacions, així com la decisió de quina o quines operacions aplicar està descrita a l’apartat 14.5.

Figura 10: Exemples de diferents túnels complexos, tots amb els mateixos paràmetres, i amb 100 extrusions. Es poden observar els diferents resultats a cada execució

D’aquesta forma, es genera un túnel amb diferents formes més complexes que a l’apartat anterior. En aquest cas, degut als canvis de direcció, caldría evitar les interseccions entre les diferents parts del túnel, explicat a l’apartat 14.10.

Per tant, el que s’estaria fent en aquest apartat seria una generació procedural de models de túnels.

12.3 Cova

A partir de l’extrusió d’una polilínia durant la generació d’un túnel, es pot decidir marcar uns quants vèrtexs de la polilínia original i la creada a l’extrusió per tal de generar un forat format per aquests vèrtexs. Els vèrtexs escollits sempre seran els mateixos per a la polilínia original i la polilínia extrudida, per motius de simplicitat i per evitar la generació d’artifacts. Aquest forat generat es pot interpretar com a l’entrada d’un nou túnel complex,
i això és el que s’utilitzarà precisament per a poder generar la cova: realment serà un conjunt de túneles, els quals seran tractats en diferent ordre segons l’ordre de la creació de la seva entrada, és a dir, del forat a partir del qual comencen. Per tant, existeixen diverses versions de l’algoritme que únicament difereixen en aquest ordre de generar els túneles, la qual cosa s’explica amb més detall a l’apartat 13.

Figura 11: Exemples de diferents coves generades vistes des de l’exterior, totes amb els mateixos paràmetres, i amb 100 forats com a màxim. Es poden observar els diferents resultats a cada execució

S’evita també la intersecció entre els diferents túneles, explicat de nou a l’apartat 14.10.
13 Versions

En aquest apartat es pretén donar una visió general del procés de generació en forma de pseudocodi, separant segons les diferents versions implementades. No es pretén entrar en detalls d’implementació, ja que això es farà als apartats següents.

Aquestes versions es diferencien essencialment en la forma en la que es tracen els túnels o forats generats. És a dir, cada cop que es genera un forat o bifurcació a una extrusió, es genera un túnel el qual necessita ser extrudit a la vegada. L’ordre en el qual es decideix quin túnel s’extrudeix abans és el que permet diferenciar entre les versions implementades, tenint una forma final de la cova per als tres casos.

En quant al cost computacional total de la generació, és el mateix per a les tres versions tal com es justifica a l’apartat 14.10. Aquest cost és en cas pitjor i molt extrem de $O(n \cdot (\text{maxExtrudeTimes} \cdot \text{maxHoles})^2)$. Els valors de l’entrada seran els mateixos per les tres versions i representen el número de vèrtexs de les polilínies, el número màxim d’extrusions per túnels i el número màxim de túnels que es poden generar, respectivament. Aquest cost no es pot aproximar segons la versió, ja que depèn dels valors generats aleatoriament. Per tant, s’utilitza la mateixa fita superior per als tres casos degut a depèn de sempre dels valors anomenats, sense ser més costós una versió que una altra.

Les tres versions reben com a paràmetre la polilínia que marca l’inici del túnel, així com un valor probabilístic que el generador d’operacions fa servir per decidir la creació d’un forat o no.

S’utilitzen dos valors parametrizables per l’usuari: el número màxim de túnels que es poden crear (maxHoles) i el número màxim d’extrusions (maxExtrudeTimes) per túnel. Inicialment es va valorar el anar variant el valor del número màxims d’extrusions per cada túnel, per tal de què hi hagués més irregularitat. Tot i així, finalment s’ha deixat com un valor estàtic ja que per motius d’interseccions aquest valor ja es variava indirectament.
13.1 Recursiva

Es produirà una crida recursiva per cada túnel, de forma que cada cop que es faci un forat es comencera la generació del nou túnel a partir del forat creat.

D’aquesta forma, per a cada túnel, s’utilitzarà un bucle per a controlar les extrusions, generant noves operacions quan calgui segons el generador de decisions:

**Pre:** Rep la polilínia inicial del túnel suavitzada i sense inicialitzar; junt amb la probabilitat de fer un forat

**Post:** un túnel i tots els túneles creats per aquest

GenerarRecursiu(polOriginal, probForat)

Incrementar comptador Forats
Inicialitzar Túnel
Mentre no s’arribi al número màxim d’extrusions
polNova = Extrudir(polOriginal, operacions)
Si operació conté forat
forat = Crear forat(polOriginal, polNova)
GenerarRecursiu(forat, probForatDecrementat)
Afegir polNova per Interseccions
polOriginal = polNova
Actualitzar operacions
Tancar túnel
Actualitzar informació Interseccions

Figura 12: Exemples de coves generades a diferents execucions utilitzant la versió recursiva, sempre amb els mateixos paràmetres

Es pot assegurar la finalització de crides recursives ja que per decidir si fer
un forat o no es té en compte el número de forats fets fins al moment, tenint un límit màxim de forats com a paràmetre.

### 13.2 Iterativa

Aquesta versió, totalment al contrari que la recursiva, s’espera a la finalització de la creació d’un túnel abans de començar a extrudir els forats generats a aquest. Per tant, a mesura que es van creant els diferents forats, aquests són guardats a una estructura de dades que és la que permet diferenciar una versió iterativa d’una altra.

El pseudocodi de la versió iterativa és el següent:

**Pre:** Rep la polilinia inicial del túnel suavitzada i sense inicialitzar; junt amb la probabilitat de fer un forat

**Post:** Genera un túnel i insereix a l’estructura de dades els forats generats

`GenerarIteratiu(polOriginal, probForat)`

1. Inicialitzar Estructura de dades
2. Mentre hagi túneles per tractar (ED)
   1. `polOriginal = Obtenir informació de túnel actual(ED)`
   2. Inicialitzar Túnel
   3. Mentre no s’arribi al número màxim d’extrusions
      1. `polNova = Extrudir(polOriginal, operacions)`
      2. Si operació conté forat
         1. `forat = Crear forat(polOriginal, polNova)`
         2. Incrementar comptador de forats
         3. Afegir forat a ED
      4. Afegir `polNova` per Interseccions
      5. `polOriginal = polNova`
   4. Actualitzar operacions
5. Tancar túnel
6. Actualitzar informació Interseccions
7. Decrementar `probForat`
13.2.1 LIFO
Per aquesta versió s’utilitza una pila per tractar els forats, de forma que al acabar un túnel es visitaran els forats en ordre invers de la seva creació, de forma que es visitarà primer l’últim forat creat.

Figura 13: Exemples de coves generades a diferents execucions utilitzant la versió iterativa amb piles, sempre amb els mateixos paràmetres

13.2.2 FIFO
Per aquesta versió s’utilitza una cua per tractar els forats, de forma que al acabar un túnel es visitaran els forats creats en l’ordre en el que s’han creat.

Figura 14: Exemples de coves generades a diferents execucions utilitzant la versió iterativa amb cues, sempre amb els mateixos paràmetres. Es pot observar com a les versions iteratives es generen coves més grans que a la recursiva, per raons d’intersecció
14 Generació detallada

En aquesta secció, es detallaran tots els diferents apartats que intervenen en la generació de la cova, explicant detalls tècnics així com aspectes importants de la implementació i les decisions presses.

14.1 Polilínia

Per definició, una polilínia és un objecte geomètric el qual és la unió de diferents vèrtexs mitjançant segments o arestes entre ells. Per aquest projecte, totes les polilínies compartiràn les mateixes propietats, degut a què serà el nucli clau per a tot l’algoritme al estar contínuament sent utilitzat:

1. Sempre serà tancada, això vol dir que el primer vèrtex serà igual a l’últim en quant a posició. Hi hauran altres aspectes com les coordenades de textura que podran variar per a dos vèrtexs d’una mateixa polilínia amb la mateixa posició.

2. Cada vèrtex de la polilínia tindrà exactament dos vèrtexs veïns, que seran referits com el següent i l’anterior vèrtex.

3. Els vèrtexs estaran ordenats en sentit horari, i seguint un sistema de la mà esquerra com es justifica a l’apartat 14.8.

Figura 15: Polilínia que compleix totes les propietats: tancada, dos vèrtexs veïns, sentit horari, vèrtexs coplanaris, convexitat, simplicitat i n = 10, suposant que el radi està entre el mínim i el màxim

5. Complirà la condició de **convexitat**, la qual vol dir que des de qualsevol punt interior a la polilínia, el segment que forma amb qualsevol vèrtex no intersecciona amb cap aresta de la polilínia. Per tal de comprovar si una polilínia és convexa, s’adaptarà de la forma que es fa en 2D: per cada vèrtex es comprova l’orientació mitjançant el producte vectorial entre el vector format pel vèrtex anterior i l’actual i entre el vector formar pel vèrtex actual i el següent. Per tal de què sigui convex, totes les orientacions han de ser les mateixes, és a dir, els resultats dels productes vectorials han de ser o bé tots positius o bé tots negatius. Per al cas de 3D, aquesta orientació es mira depenent si queda per davant o per darrere del pla que forma la polilínia. Aquesta comprovació té una complexitat lineal en el nombre de vèrtexs, que serà l’entraida principal respecto la qual es faran tots el costos.

6. Serà **simple**, el que vol dir que cap aresta d’una mateixa polilínia es podrà creuar amb una altra. Inicialment es van estudiar diferents algorithmes per comprovar que una polilínia fos simple, com per exemple l’algoritme de Bentley-Ottmann o el de Shamos-Hoey[24], ambdós amb cost $O(n \cdot \log n)$ on $n$ és el número de vèrtexs de la polilínia; o fins i tot la prova de tots els vèrtexs amb tots, amb cost $O(n^2)$. Tot i així, degut a què convexitat implica simplicitat per definició, realment no cal fer comprovacions explícites de si una polilínia és simple o no un cop s’ha comprovat si és convexa.

7. Totes les polilínees pertanyents a extrusions tindran el mateix número de vèrtexs, pel que la **mida de totes les polilínees serà constant i igual**.

8. Totes les polilínees tenen un **radi mínim i màxim**, per tal d’evitar polilínees massa petites o massa grans, el qual tindrà el valor constant 2.0 i 40.0 respectivament. Per fer la comprovació del radi mínim es calcularà amb la distància mínima de tots els vèrtexs de la polilínia i el baricentre, mentre que per al radi màxim serà la distància màxima.
14.2 Entrada de cova

Primer de tot, caldrà decidir quina forma aproximada tindrà l’entrada de la cova. Per tal de fer això, es podrà seleccionar el número de punts que s’utilitzaran per generar la forma de l’entrada de cada túnel, creant així una polilínies inicial a partir de la qual es començarà tota la generació. Els punts hauran de ser escollits en sentit horari degut al que es comentà a la secció 14.8. L’usuari podrà veure els punts seleccionats ja que es dibuixen segments de rectes entre ells per tal de millorar la interacció.

El número de punts a triar és decidit pel paràmetre **Gate Size**.

Figura 16: Exemples de polilínies on no s’infringeixen algunes de les propietats esmentades

\[\text{(a) No convex però si simple} \quad \text{(b) No simple i per no tant no convex}\]

\[\text{Figura 17: A partir de pocs punts seleccionats per l’usuari, es genera una coa bastant suavitzada, tot i perdre espai escollit inicialment}\]
Aquesta polilínia inicial inicialment tenia tants punts com vèrtex necessaris per a una polilínia, ja que no s’aplicava cap suavitzat. La implementació d’aquest va ajudar a tenir pocs punts per tal de facilitar l’entrada d’aquests per l’usuari, per el que se li aplicarà un suavitzat per no tenir una cova massa poligonal, és a dir, amb forma molt punxeguda. Després d’aquest suavitzat ja podrà ser tractada de la mateixa forma que l’inici d’un túnel, de la mateixa forma que es fa quan es crea un forat. Tot aquest procés d’inicialització s’explica al subapartat següent.

### 14.3 Inicialitzar túnel

Cada cop que es vol començar a generar un túnel, bé perquè és l’inici de la cova o bé perquè s’ha fet un forat a un túnel ja existent, es fan alguns preparatius per al correcte funcionament del programa. Es rebrà la polilínia inicial del túnel amb el suavitzat ja fet, per tant no caldrà fer cap subdivisió ni afegir cap vèrtex nou.

Primer de tot, es crearà una nova mesh que es guardarà tota la informació generada per a poder renderitzar el túnel actual, i per tant la cova. També se li assignaran als vèrtexs corresponents d’aquesta polilínia inicial els primers índexs de vèrtex de la mesh, degut a què és l’inici d’aquesta.

Després, es crearà el conjunt d’operacions assignades per la generació d’aquell túnel, que inicialment estarà configurat per a no aplicar cap operació, i a esperar a decidir si generar cada operació diferent o no segons els paràmetres indicats per l’usuari.

Finalment, i abans de començar a extrudir la polilínia inicial, s’haurà d’indicar la direCCIó en la qual es començarà a extrudir. Per tal de calcular-la, s’utilitzarà la normal del pla format per la polilínia.

Inicialment, els vèrtexs d’una polilínia podrien no formar un pla a l’espai 3D, pel que no es podia calcular la normal d’un pla al no tenir cap real i aquest s’havia d’aproximar. No es podia calcular la normal directament amb el producte vectorial degut a què es depèn molt de quines dos arestes s’agafen, ja que podrien no ser coplànàries o bé ser quasi colineals i generar valors numèrics inestables.

---

8 La informació corresponent generada, així com la seva utilitat s’especifica a la secció 14.9

9 La decisió de quines operacions generar, així com dels possibles paràmetres, estan està explicit a l’apartat 14.5.1
Al final, però, per raons d’una millor visualització de la cova tots els vèrtexs d’una mateixa polilínia formen un pla exacte, com s’ha vist a les propietats de la polilínia. Tot i així, degut a què va ser un dels primeres tasques a implementar per raons de necessitat, s’utilitza un mètode on a partir d’un conjunt de vèrtexs qualsevol, es calcula la normal de, com s’ha esmentat anteriorment, el pla aproximat que formen aquests vèrtexs.

Aquesta normal es calcula a partir de l’àrea del polígon resultant de projectar la polilínia 3D en els tres diferents plans coordinats \((x,y,z)\). L’àrea de cada projecció correspon a una component de la normal de la polilínia original.

Per a calcular l’àrea en 2D, l’algorisme parteix de la idea de sumar tots els paral·lelograms que \(\text{van cap a l’esquerra}\) (segons l’ordre dels vèrtexs, com el que apareix pintat de color marró), i restar les àrees dels que \(\text{van cap a la dreta}\), com el que apareix de color groc segons la figura 18.

\[
A(P) = \frac{1}{2} \sum_{i=0}^{n-1} (y_i + y_{i+1})(x_i - x_{i+1})
\]

\(a\) on \(i \oplus 1 \equiv i + 1 \text{ (mod } n)\)

Figura 18: Càlcul de l’àrea d’una polilínia en 2D

Per tant, per tal de calcular la normal en 3D es pot adaptar i obtenir la següent expressió per a cada component del vector resultant.

\[
\begin{align*}
n_x &= \frac{1}{2} \sum_{i=0}^{n-1} (z_i + z_j)(y_i - y_j) \\
n_y &= \frac{1}{2} \sum_{i=0}^{n-1} (x_i + x_j)(z_i - z_j) \\
n_z &= \frac{1}{2} \sum_{i=0}^{n-1} (y_i + y_j)(x_i - x_j)
\end{align*}
\]

\(a\) on premem \(j \equiv i + 1 \text{(mod } n)\)

Figura 19: Càlcul de la normal d’una polilínia qualsevol

Finalment, caldrà normalitzar el resultat, ja que la llargada resultant és igual
a l’àrea ocupada per la polilínia en tres dimensions. També caldrà negar el resultat ja que les fórmules esmentades tracten els vèrtexs en ordre antihorari, mentre que els vèrtexs de les polilínies que s’utilitzen a la generació sempre estaran en ordre horari per definició.

### 14.4 Extrusió

L’extrusió és una operació on, a partir d’un conjunt de vèrtexs que formen una polilínia, es generen un nou conjunt de vèrtexs de la mateixa mida a una certa distància i seguint una determinada direcció, formant així una nova polilínia. Bàsicament vindria a ser sumar un mateix vector a un conjunt de punts

![Imatge de l’extrusió](image)

Figura 20: Exemple de com es forma una extrusió: a partir de la polilínia $a$ (verda), es generarà la polilínia $b$ (blava), extrudint tots els vèrtexs en la mateixa direcció (negre)

Abans d’afegir la nova polilínia a la malla de triangles que representen la cova, cal comprovar que l’extrusió actual no estigui interseccionant amb parts de la cova ja creades, ja que sinó s’obtindrien uns resultats no desitjables. Per tant, primer s’extrudirà la polilínia actual creant una nova, aplicant les operacions que toquin. És mantindrà una propietat que, degut a extrudir en ordre, **el vèrtex i-èssim de la nova polilínia sempre serà generat a partir del vèrtex i-èssim de la polilínia original**, de forma que sempre estaran units. Després d’això, es comprovarà la intersecció amb la cova ja creada, i en cas de què no es produeixi cap, s’afegirà aquesta nova polilínia a la malla, de forma que s’afegiran tots els seus vèrtexs, i més tard els triangles de l’extrusió actual.
correspondents que s’hagin de triangular. No totes les extrusions tindran una
triangulació completa degut als forats, ja que la part de l’extrusió corresponent
a un forat ens interessa deixar-la sense triangular.
Per aquest motiu, després d’haver generat la nova polilínia i abans de triangular
l’extrusió, es comprovarà mitjançant l’operació actual si cal fer un forat. En cas
positiu, es cridarà a una funció que decideix on fer el forat\(^\text{10}\). Pot ser que per
raons les quals no s’entrarà en detall en aquesta part de la memòria, no sigui
possible fer un forat. En aquest cas, caldrà reextrudir i generar una nova po-
ilínia amb una distància petita, degut a què la distància d’extrusió és més gran
en cas de què s’hagi de fer un forat.
Després de mirar els forats, caldrà també comprovar si s’han de generar esta-
lagnites, estalactites o pilars en l’extrusió actual.

Gràcies a tenir el codi estructurat d’aquesta forma, en el cas de què en el
futur es volgués expandir el projecte afegint-hi noves funcionalitats, només cal-
dria afegir després del que s’ha explicat la comprovació de si caldría fer aquesta
nova operació, i en cas positiu, aplicar-la.

Així doncs, el pseudocodi d’una extrusió és el següent, que vindria a ser
l’ampliació del pseudocodi de l’algoritme general:

\textbf{Pre: Rep una polilínia juntament amb les operacions a aplicar}
\textbf{Post: Crea una nova polilínia a partir dels paràmetres i
actualitza les operacions per la següent extrusió}

\textbf{Extrudir}(polOriginal, operacio)

\hspace*{1cm} \text{Distancia} = \text{operacio.obtenirDistancia}
\hspace*{1cm} \text{Direcció} = \text{operacio.obtenirDirecció}
\hspace*{1cm} \text{IncrementUV} = \text{ObtenirIncrementUV}
\hspace*{1cm} \text{Per cada vertex de polOriginal}
\hspace*{2cm} \text{Extrudir nou vertex a partir de vertex actual}
\hspace*{3cm} \text{PosicioNouVertex} = \text{PosicioVertexOriginal + Direcció\textbullet Distància}
\hspace*{3cm} \text{IndexNouVertex} = \text{NúmeroVertsMeshActual + NumeracioIteracio}
\hspace*{3cm} \text{CoordenadesUVNouVertex} = \text{CoordenadesVertexOriginal + IncrementUV}
\hspace*{1cm} \text{AplicarEscalat}
\hspace*{1cm} \text{AplicarRotacio}
\hspace*{1cm} \text{Si intersecció a l’extrudir}

\(^{10}\)Explotat amb més detall el funcionament dels forats a l’apartat 14.6
Sortir bucle extrusions  
Si operació conté forat  
Crear forat  
Si no s’ha pogut crear el forat  
Reextrudir amb distància petita  
Altrament  
Tractar forat  
\texttt{meshActual.afegirPol(novaPolilinia)}  
\texttt{meshActual.triangular(polOriginal, novaPolilinia)}  
Si operació conté estalagmites, estalactites o pilars  
Generar estalagmites, estalactites o pilars  
Si operació conté nova operació  
Generar noves operacions  
ActualitzarInterseccions  
ActualitzarOperacions

La línia de \textit{Generar noves operacions} és on donaria lloc a posar possibles operacions o objectes extres a la cova. Totes les altres operacions que intervenen en aquest procés són comentades amb més detall al seu subapartat corresponent.

### 14.5 Operacions

Com s’ha comentat en diversos apartats anteriors, les diferents operacions permeten aplicar canvis a la cova i són el que realment fan que hi hagi una generació procedural, pel que són una part essencial del algoritme.

Hi ha totes les possibles operacions llistades al següent subapartat, algunes de les quals s’expliquen amb més detall a altres apartats de la memòria, degut a la seva complexitat.

Inicialment només es generava una operació a aplicar, i quan s’havia acabat d’aplicar es generava la següent, triant un número aleatori entre totes les possibles operacions per tal de decidir quina aplicar.

Més tard, es va sofisticar de forma que es poguessin aplicar més d’una operació a la mateixa extrusió, de forma que cadascuna de les operacions excepte els
Aquests paràmetres es tenen en compte a cada moment que es crida al generador d’operacions, el qual és sempre després d’una extrusió. D’aquesta forma es permet que la següent extrusió ja sàpiga quines operacions cal aplicar.

Per tal de generar una nova operació o no, es consultarà l’estat de les operacions actuals, seguint el mateix patró per cada una de les operacions, exceptuant la forma en la qual es decideix com generar forats. Així doncs, el pseudocodi de com es generà una operació qualsevol excepte forats seria el següent:

Pre: Es rep una operació
Post: Genera els valors necessaris sobre com aplicar l’operació rebuda durant les següents extrusions
Si operació ha de generar-se
  duracioExtrusio = generarRang(opExtrBase, opExtrDesv)
  duracioEspera = generarRang(opKBase, opKDesv)
  valor = operació.generarValorOperacio
  operació.ExtrudirDurant(duracioExtrusio, valor)
  operació.EsperarDurant(duracioEspera+duracioExtrusio)

Per saber si una operació necessita generar-se es comprova si un comptador el qual es va decrementant a cada extrusió és 0. Això vol dir que quan arribi a 0 s’haurà de generar els valors de l’operació corresponent i aplicar-la durant el número d’extrusions marcades pel valor duracioExtrusio generat aleatòriament, així com posar aquest comptador a duracioEspera + duracioExtrusio, de forma que passades aquest número d’extrusions es torni a generar l’operació. Al inicialitzar un túnel, aquest comptador s’inicialitza amb un valor equivalent a duracioEspera.

Aquesta forma de generar les operacions permet molt control de la freqüència de cada operació des de el punt de vista de l’usuari, ja que només cal variar els
quatre paràmetres esmentats per a cada operació.

De forma lògica, \textit{generarValorOperacio}, varia per a cada operació, al subapartat següent es comenta la forma de generar cadascuna, tot i que sempre hi ha la generació de valors aleatoris pel mig.

\textbf{14.5.1 Conjunt d’operacions}

A continuació s’esmenta el valor que es genera per a cada una de les diferents operacions que es poden aplicar.

- **Canvi de direcció**: Es genera una direcció aleatòria amb la \( y \) limitada per el paràmetre \textit{directionYWalkLimit}, i es comprova que l’angle entre aquesta direcció i la direcció actual d’extrusió sigui menor al paràmetre \textit{directionMaxAngle}. Si no es compleix aquesta restricció amb l’angle, es tornarà a generar una nova direcció, i així successivament fins que es trobi una direcció que compleixi la restricció de l’angle, o bé s’hagin fet un número d’ intents amb un valor constant per tal de no quedar-se en bucle infinit. Aquest angle pot prendre un valor de fins a 40\(^\circ\), ja que a partir d’aquest valor es considera un canvi massa brusc, i pot provocar la reducció del radi de la polilínia de forma massa exagerada d’una extrusió a una altra, acabant fins i tot produint \textit{artifacts}.

Es va plantejar el idear un mètode que fos menys aleatori per tal de poder arribar a evitar les interseccions i trobar un camí pel que es pogués extrudir, sense haver de parar la generació del túnel actual per culpa d’una intersecció. Tot i així, es va considerar que aquesta comprovació era massa costosa, pel que es va descartar.

Figura 21: Exemple d’aplicar constantment l’operació de canvi de direcció
• **Canvi de distància:** Hi ha dos possibles distancies: les distàncies grans i distàncies petites. El primer tipus s’utilitza per quan s’ha de crear un forat, mentre que el segon per tots els altres tipus d’extrusions. Els dos són generats a partir d’un número aleatori generat entre un rang parametrizable: 

\[
\text{distanceBigMin}, \text{distanceBigMax}
\]

per les distàncies grans i

\[
\text{distanceSmallMin}, \text{distanceSmallMax}
\]

per les petites. S’ha de vigilar amb no ficar distàncies mot petites per les segones ja que això provoca més interseccions i per tant que quedarà una cova més petita degut al mètode que es fa servir per evitar les interseccions; però tampoc massa gran ja que això provocaria pèrdua de resolució i coves massa grans. La duració d’aquesta operació dura només una extrusió, al estar constantment generant nous valors de distància a cada extrusió per tal d’haver-hi més irregularitat.

• **Escalat:** Es generarà un valor aleatori pertanyent al rang parametrizable

\[
[1.15-\text{scaleLimit}, 1.15 \text{scaleLimit}]
\]

, amb la restricció de què \text{scaleLimit} no pot ser superior a 0.99 per no tenir escalats massa petits. Per la mateixa raó s’ha triat el valor 1.15 en comptes de 1. A partir del valor d’escalat, cada vèrtex de la polilínia s’escalarà prenent com a centre el baricentre de la polilínia. Abans d’escalar, però, caldrà comprovar que la polilínia resultant d’aplicar l’operació segueixi complint la propietat de radi mínim i màxim. Com que s’ha de fer l’escalat per a cada vèrtex i per a cada vèrtex l’escalat és una operació constant, es té un cost total \(\theta(n)\), on \(n\) és el número de vèrtexs de la polilínia.

Figura 22: Exemple d’aplicar constantment l’operació d’escalat

• **Rotació:** Es genera un valor aleatori que pertany al rang parametrizable

\[
-\text{rotationLimit}, +\text{rotationLimit}
\].

62
Per tal d’aplicar l’operació de rotació a una polilínia, primer de tot s’obtindrà el seu baricentre, i es rotarà cada vèrtex de la polilínia al voltant d’aquest en la direcció d’extrusió, tants angles com indiqui el valor de rotació. El cost d’aquesta operació per tant és de $\theta(n)$.

- **Forat**: Tot i que no es genera el valor de la mateixa forma que de les altres operacions, l’operació de fer un forat també s’ha de guardar, i ho fa en forma de booleà. D’aquesta forma, quan sigui cert a una extrusió, es farà el forat si es compleixen les condicions i després es posarà a fals per no estar generant forats constantment. La generació d’aquest booleà s’estarà comprovant a cada extrusió, i tindrà en compte tres paràmetres: el número d’extrusions del túnel actual, la probabilitat de fer forat per al túnel actual la qual es va decreixent a mesura que s’està en un túnel més profund, i el número total de forats fets. Existiran doncs diverses condicions que utilitzen aquests paràmetres per decidir si generar un forat a la següent extrusió o no.

Independentment de la condició, sempre es comprovarà a l’inici que el número d’extrusions sigui menor a un valor petit per tal de no generar un forat massa al principi d’un túnel. També es comprovarà que el número de forats create sigui menor al permès per la generació actual, especificat pel valor parametrizable `maxHoles`. La condició serà la mateixa per tota la generació i es pot parametrizar mitjançant la variable `holeCondition`:

- EachKDesv: Cada K extrusions s’intentarà generar un forat. Aquesta K és el valor parametrizable `holeK`
- EachKDesvProb: Serà similar a l’anterior, però en comptes d’intentar generar un nou forat segur, existirà una probabilitat de fer-ho. Aquesta probabilitat correspon al valor parametrizable `holeProb`.
- MoreExtrMoreProb: A mesura que el túnel actual porti més extrusions, existirà més probabilitat d’intentar fer un forat. Utilitzarà el valor `holeProb`, juntament amb el valor `holeLambda`, que rebra un valor petit. D’aquesta forma, la probabilitat d’intentar fer un forat a cada extrusió serà de `holeProb + numExtrusions*holeLambda`.
- MoreExtrLessProb: És el cas contrari a l’anterior, on a mesura que es va fent extrusions hi ha menys probabilitat de fer un forat, de forma que aquesta probabilitat és de `holeProb - numExtrusions*holeLambda`.

63
Aquestes diverses condicions es van crear a l'inici del projecte per veure com podia afectar cadascun als forats generats. A mesura que s'anava avançant, es va observar que la millor condició era el mètode EachK, amb un valor de K petit. Això no generà tants forats com es podria esperar ja que per tal de generar un forat s'han de complir unes certes condicions que tenen una dependència directa en com s'ha format la cova fins al moment. D'aquesta forma, cal estar provant de forma bastant regular si es pot generar un nou forat.

- **Il·luminació:** Es genera un booleà indicant si cal generar un punt de llum o no, i es generarà sempre després s'haver transcorregut el número d'extrusions d'espera, pel que la seva generació només depèn de generarRang (opExtrBase, opExtrDesv). La duració d'aquesta operació es d'una extrusió. Inicialment la il·luminació es generava d'aquesta forma, però al veure resultats que no acabaven de convèncer es va acabar eliminant aquesta operació, generant l'il·luminació amb un altre mètode\(^{11}\).

- **Estalagmites:** Es genera un objecte del tipus estalagmita, estalactita o pilar, mitjançant la generació d'un valor aleatori entre un rang qualsevol, donant una probabilitat de 40 40 20 per generar o bé estalagmites, estalactites o un pilar amb estalagmites i pilars respectivament. La duració d'aquesta extrusió ve marcada pel rang aleatori generat.

- **Noves operacions:** A partir d'aquí, es dóna la llibertat a un altre usuari programador que pugui crear qualsevol altra operació, ja sigui per manipular els vèrtexs de l'extrusió o bé poder afegir nous elements.

En algunes ocasions, el valor generat no serà directament el que s'apliqui a cada operació, sinó que serà una modificació d'aquest per tal d'obtenir uns resultats suavitzats\(^{12}\).

És molt important destacar que cap de les operacions implementades variaran la polilínia de forma que es deïxi de complir alguna de les seves propietats.

---

\(^{11}\)Tal i com es comenta a l'apartat 14.13

\(^{12}\)Així doncs, les operacions que no utilitzen directament el valor generat es comenten a l'apartat 14.11
14.5.2 Generació valors aleatoris

Sempre que es vol generar un número aleatori, s’utilitza la funció del propi Unity, el qual reallyment genera un valor pseudoaleatori fent servir el que s’anomenen seeds, o llavors[25]. A partir d’aquesta llavor, es generen sempre el mateix conjunt de números aleatoris, de forma que a partir de l'últim generat, cada cop que s’està cridant a la funció d’obtenir un nombre aleatori, es genera un nou nombre. És important destacar que sempre es generarà a partir de l’últim i sempre serà el mateix. Aquest procediment rep el nom de Random Number Generator(RNG), que és una tècnica molt utilitzada a videojocs per generar qualsevol valor aleatori, des de generacions procedural o per exemple els objectes que pot deixar caure un enemic.

D’aquesta forma, amb el mateix valor de llavor s’obtindrà el mateix resultat, cosa que pot ajudar molt a repetir una generació. Per defecte, si a Unity no se li especifica cap llavor, crearà una de nova a partir de la data actual.

A part d’aquest mètode es va estudiar fer mètodes propis de generació, com per exemple l’opció de què el rang dels possibles valors a generar poguessin seguir diferents distribucions probabilístiques, com per exemple una Gaussiana, una Exponencial, ... Finalment, però, ja s’obtenia un resultat força bo utilitzant directament la funció de Unity. A més a més, utilitzar diferents distribucions probabilístiques hauria complicat la possibilitat de què es pogués tenir un fàcil control de tot el relacionat amb la generació d’operacions des de el punt de vista de l’usuari.

14.6 Forats

Els forats són una característica essencial per l’algoritme per tal de poder crear i unir diferents túneles.

Per tal de crear un forat, s’utilitzaran la polilínia original i la generada en una extrusió, seleccionant el mateix número de vèrtexs de cada una i corresponents entre les polilínies. És a dir, si es marca el vèrtex i-èssim de la polilínia original per a pertanyer a un forat, també es marcarà el vèrtex i-èssim de la polilínia creada a l’extrusió. Aquests vèrtexs es marcaran com pertanyents a un forat, de forma que no es triangularan entre ells i donaran peu a l’entrada d’un nou túnel, sent ahora la connexió entre el túnel actual i el túnel just generat.

Inicialment es feia un forat agafant un número de vèrtexs aleatoris seguits
a partir d’un dels vèrtexs de la polilínia, també escollit aleatòriament. Això provocava que la cova pogués tenir forats a qualsevol part i en qualsevol direcció, complicant així el moviment i l’exploració a través d’aquesta. D’aquesta manera també es generava una cova més petita per motius del mètode triat d’evitar interseccions\textsuperscript{13}.

Per tal de millorar això, es va decidir generar una direcció aleatòria pertanyent a un cert rang parametrizable per l’usuari amb el valor \texttt{directionYWalkLimit}, de forma que la direcció de polilínia (la normal) corresponent al forat, no s’allunyes massa d’aquesta direcció. Es fa tenint en compte només la component y, de forma que no es permetrà que la cova vagi massa cap abaix o massa cap amunt segons els paràmetres; tenint així una cova explorable per personatges que només puguin caminar. Per suposat que es pot parametrizar aquest valor de forma que realmente no hi hagi aquesta restricció de la direcció, però no es gens recomanable ja que com a resultat es tindrien els problemes just anomenats anteriorment.

A les primeres etapes del projecte, la distància entre una extrusió normal i una extrusió on es produïa un forat eren molt properes i petites. Això provocava que els forats generats fossin molt petits i no donessin la impressió de ser una bifurcació de la cova. Degut a què es volia que les extrusions normals continuessin sent de una distància no massa gran\textsuperscript{14}, es va imposar que el generador

\textsuperscript{13}Explicat amb més detall a l’apartat 14.10
\textsuperscript{14}Per les raons que s’expliquen a l’apartat 14.11
d’operacions generés una distància més gran per a una extrusió amb forat.
Finalment, si aquest forat no es podia crear, per tal de no tenir una extrusió
massa diferents de les altres, ja que seria més gran quan no hauria, es fa un
backtrack de l’extrusió, creant-la de nou amb distància més petita.

A continuació s’expliquen tots els passos que es segueixen per generar un
forat i per tant un túnel nou:

1. Extrudir amb distància gran, un valor aleatori pertanyent a un rang de dos
númbers parametritzables per l’usuari amb les variables distanceBigMin
i distanceBigMax.

2. Seleccionar els vèrtexs de l’extrusió que formaran el forat. Aquest pas
aprofitarà la propietat que tenen les polilínies de tenir els vèrtexs ordenats
en sentit horari, així com la propietat de l’extrusió de la correspondència
entre vèrtexs. D’aquesta forma, caldrà només trobar un índex inicial i la
mida del forat.
Inicialment per tal de trobar aquests valors, primer es generava una direc-
ció aleatòria de la mateixa forma que quan es genera un canvi de direcció:
es seleccionaven valors completament aleatoris per les components x i z
(entre -1 i 1), i un valor aleatori per la component y entre uns valors pa-
rametrizables per l’usuari amb la variable directionYWalkLimit, degut
da la restricció de poder caminar per la cova.
Més tard, es va implementar un nou mètode degut a què tanta aleatori-
etat provoca direccions massa allunyades i per tant no permeses per les
derients restriccions. Aquest mètode el que fa és primer de tot decidir si
el forat es generarà a la dreta o a l’esquerra respecte la direcció d’extrusió,
amb probabilitat 50 50. Es generarà una direcció normal aproximada del
forat a partir del producte vectorial entre la direcció d’extrusió actual i el
vector up, intercanviant l’ordre del producte segons sigui cap a la dreta o
l’esquerra. A partir d’aquesta direcció, es modificarà de forma aleatòria i
petita les tres components del vector direcció, comprovant sempre que la
component y no s’allunyi massa de la restricció comentada.

Un cop generada aquesta direcció, es compararan la direcció entre el baricentre i cada vèrtex, seleccionant com a vàlids per al forat els vèrtexs els
quals produeixin una direcció no massa allunyada a la direcció generada.
Aquesta proximitat entre direccions es fa mitjançant un límit d’angle entre direccions, sent aquest un altre valor parametrizable mitjançant la variable **holesMaxAngleDirection**. Tot i així, aquesta diferència màxima d’angle no ha de ser gaire gran, ja que de ser-ho, es podrien produir **artifacts**, tenint com a resultat una triangulació no desitjable.

Inicialment degut a què el número de vèrtexs de la polilínia era totalment aleatori, en moltes ocasions s’agafava un número de vèrtexs tan grans, que la direcció de la polilínia feia que a l’extrudir s’interseccionés amb ella mateix i es creessin **artifacts**.

![Figura 24: Es pot observar com agafar molts vèrtexs d’una polilínia al fer un forat genera una normal on en alguns casos (línies vermelles) es generaran nous vèrtexs entremig de la polilínia existent, creant d’aquesta forma artifacts](image)

Es comprovarà que el número de vèrtexs vàlids oscil·lí entre el rang parametrizable per les variables **holeMinVertices** i **holeMaxVertices**. Amb això s’haurà trobat els vèrtexs de les dues polilínies que formaran el forat, donant una polilínia simètrica (és a dir, el mateix nombre de vèrtexs per a cada polilínia i units entre ells) degut a com es generen les extrusions.

3. Per al següent pas inicialment es comprovava que la nova polilínia formada, anomenada **polilínia forat** a partir d’ara, no tingués una direcció d’extrusió inicial (una normal), amb una inclinació massa cap abaix o massa cap amunt. Això es feia comprovant la component y amb un valor límit,
el qual és parametrizable per l’usuari amb la variable \textbf{directionYWalk-Limit}. També es volia comprovar que no es creés cap \textit{artifact}. També molt al principi del desenvolupament, la comprovació de la direcció es feia un cop acabat tot el procés de creació del forat, un cop s’anés a extrudir. Això provocava que s’hagués de tancar immediatament el túnel i es quedés nombros amb una polilínia, cosa que no era gens desitjable. Per això, es va posar aquesta comprovació mentre es generava el forat.

Tot i així, amb la generació de la direcció final que s’ha fet, ja no cal comprovar que la direcció de la polilínia sobrepassi aquest límit, ja que es fa de forma implícita al generar la direcció.

4. Inicialment amb això ja s’obtenia l’inici del nou túnel. Es va observar, però, que s’obtenien resultats massa quadrats i amb formes estranyes als costats. Això era degut bàsicament a què per culpa de l’extrusió, els vàrtexs de la polilínia forat sempre quedaven molt dividits entre els vàrtexs de la polilínia extrudida i la polilínia entre una aresta molt allargada, com a resultat de l’extrusió amb gran distància. Això també podia formar \textit{artifacts} a la llarga, després d’acumular formes estranyes durant les extrusions i forats següents.

\begin{center}
\includegraphics[width=0.5\textwidth]{figura25.png}
\end{center}

\textbf{Figura 25: Generació de formes irregulars al seleccionar un conjunt de vàrtexs al fer un forat a una extrusió entre a (verd) i b (blau). Al no ser els vàrtexs coplanars i extrudir-los en direcció a la normal de la polilínia, s’acabarien produint polilínies massa irregulars, i la possibilitat de què acabessin sent \textit{artifacts}}

Per tal de solucionar això, es crea una nova polilínia a partir de la polilínia
forat, la qual serà tractada realment com el inici del túnel. Aquesta nova polilínia serà creada a partir de una projecció de la polilínia forat en la direcció de la seva normal.

Així doncs, es crearà un pla en direcció de la normal i a una certa distància de forma que no interseccioni amb la polilínia forat. La projecció inicialment només estarà formada per un subconjunt dels vèrtexs de la polilínia forat projectats. La mida d’aquest subconjunt de vèrtexs serà sempre el mateix que el número de vèrtexs utilitzats per seleccionar l’entrada de la cova. D’aquesta forma, un cop projectats seran suavitzats i escalats a la mida del forat per formar així la polilínia entrada del nou túnel, complint així la propietat de què totes les polilínies tenen la mateixa mida.

S’ha escollit aquest procediment en comptes de simplement projectar tots els vèrtexs de la polilínia forat ja que realment no caldrien tants vèrtexs, a part del problema d’estar molt dividits com s’ha comentat prèviament.

Figura 26: Projecció de la polilínia forat (verd) per crear la nova polilínia projecció (vermell) que representa l’inici del túnel creat a partir del forat

5. Un cop generat aquesta nova polilínia, anomenem-li **polilínia projecció**, és quan s’ha de fer la comprovació d’evitar que no es generi cap artifact comprovant que la polilínia projecció sigui convexa i simple. Per definició, basta amb comprovar que sigui convexa ja que això implica simplicitat.

També es fa la comprovació de què la polilínia projecció no tingui un radi menor al permés per a qualsevol polilínia, així com la de què no interseccioni amb cap altre part de la cova ja generada.

6. Cal inicialitzar els índexs de la polilínia projecció, així com les seves co-
ordenades de textures, comentat en el respectiu apartat. Finalment, cal triangular entre la polilínia forat i la polilínia projecció. Per tal de fer-ho, es seguirà un mètode que beu del càlcul d’envolupants convexes en 3D\textsuperscript{15}.

Per tant, el pseudocodi de la generació d’un forat seria el següent:

Pre: Rep dos polilínies, on la segona ha estat generada a partir d’extrudir la primera
Post: Retorna la polilínia ja suavitzada que permet crear un nou túnel a partir d’un forat generat, o null si no s’ha pogut crear el forat

CrearForat(polOriginal, polNova)
   polHole = SeleccionarCandidatsForat(polOriginal,polNova)
   Si polHole es null
      Retornar null
   polProjection = polHole.projectar
   Si !polProjection.esConvexa || polProjection.midaIncorrecte || polProjection.produeixInterseccio
      polOriginal.desferForats
      polNova.desferForats
      Retornar null
   polProjection.inicialitzar
   meshActual.afeigarPol(polProjection)
   meshActual.triangular(polHole,polProjection)
   Retornar polProjection

Pre: Rep dos polilínies, on la segona ha estat generada a partir d’extrudir la primera
Post: Retorna la polilínia que representa el forat creat amb vèrtexs de les polilínies paràmetres, o null si no s’ha pogut crear el forat

SeleccionarCandidatsForat(polOriginal,polNova)
   dirAprox = random.esquerra ? cross(polOriginal,up) : cross(up,polOriginal)
   dirAprox = random.canviar(dirAprox)
   primer = Obtenir primer vertex de polrOriginal que direccio amb baricentre no s’allunya molt de dirAprox
   midaForat = 0

\textsuperscript{15}Aquest mètode i les seves conseqüències està comentat amb detall al subapartat 14.8.2
Mentre direccio(polOriginal.obtenirVertex(primer+midaForat),
    baricentre) no s’allunya de dirAprox
    +++midaForat
midaForat **= 2
Si midaForat < holeMin || midaForat > holeMax
    Retornar null
i = 0
Mentre i < midaForat/2
    polOriginal.marcarForat(primer+i)
    forat.afegirVertex(polOriginal.obtenirVertex(primer+i))
i++
Mentre i > 0
    i--
    polNova.marcarForat(primer+i)
    forat.afegirVertex(polNova.obtenirVertex(primer+i))
Retornar forat

14.7 Estalagmites, estalactites i pilars

Aquest apartat explica el métode utilitzat per a la creació tant d’estalagmites (trossos de cova que surten del terra), estalactites (trossos de cova que surten del sostre) com de pilars, que realment són la unió d’una estalagmita i una estalactita.
A partir d’ara s’utilitzarà el terme espeleotema\textsuperscript{16} per referir-nos a qualsevol d’aquests tres tipus d’objectes que es crearan.
Inicinentalment es volia generar un espeleotema a partir de quatre vèrtexs veïns d’una extrusió, dos de cada polilínia.
Aquesta decisió va tenir dos canvis: El primer d’ells va ser poder generar-lo amb més vèrtexs, sempre un número parell que depengués de la distància d’extrusió de forma que la base de l’espeleotema quedés el més quadrada possible per tal de semblar més real. Tot i així, es va imposar un límit de 5 vèrtexs per polilínia, per tal de què no quedés massa exagerada
El segon, també per intentar afegir més realisme i per tant immersió, va ser el poder crear més d’un espeleotema per extrusió, degut a què normalment aquestes són creades en petits grups, degut a erosions i altres raons naturals del
\textsuperscript{16}Una paraula utilitzada per al deposit de minerals que formen diferents cavitat a les coves i altres medis semblants
comportant del medi d’una cova. D’aquesta forma, a una mateixa extrusió es podran generar com a màxim un pilar més estalagmites, o bé un pilar com a màxim més estalactites, depenent del que s’hagi decidit al generador d’operacions. A més, per tal de reforçar el concepte de què estiguin en petits grups, la duració de l’operació de generar espeleotema hauria de durar més d’una extrusió

El procés per crear-los a cada extrusió s’explica a continuació:

1. Obtenir la mitjana de distància entre vèrtexs per, a partir d’aquest valor i el de la distància d’extrusió, decidir quants vèrtexs s’utilitzaran per tal de què l’espeleotema creat tingui una base el més quadrada possible.

2. Obtenir els vèrtexs candidats a partir dels quals es crearà un espeleotema. Degut a la propietat d’estar ordenats i que totes els espeleotemes utilitzaran el mateix número de vèrtexs, només cal guardar-se el primer dels
vèrtexs que tindran cadascun d’ells.
Per tal de trobar aquests candidats, primer es decidirà quants espeleotemes es crearan per a l’extrusió actual, sent aquest un valor aleatori entre 1 i 5. Després s’iterarà tants cops com espeleotemes es vulguin crear, i per cadascuna d’aquestes iteracions es miraran tots els possibles vèrtexs a partir dels quals es pot generar l’espeleotema. Per decidir el millor candidat, es mirarà quin de tots els vèrtexs formaria la base de l’espeleotema amb la direcció (normal) més propera a la direcció objectiu. Aquesta direcció serà el vector down per les estalactites i pilars, i el vector up per a les estalagmites. Un cop s’hagi seleccionat el vèrtex candidat, es formarà la polilínia inicial de l’espeleotema a partir de la mida que ha de tenir aquest. A mesura que es vagin obtenint els candidats, caldrà marcar com a invàlids els vèrtexs que ja s’han obtingut, de forma que els espeleotemes no interseccionin entre ells. El pseudocodi per obtenir aquests vèrtexs candidats seria el següent:

\[
\text{Pre: Rep dos polilínies, on la segona ha estat generada a partir d’extrudir la primera; juntament amb la mida i direcció desitjada de l’espeleotema}
\]
\[
\text{Post: Retorna els vèrtexs inicial a les polilínies amb direcció de la normal més propera a la desitjada per formar un espeleotema entre les polilínies paràmetre i amb la mida rebuda segons el paràmetres obtinguts}
\]
\[
\text{ObtenirCandidatsEspeleotema(sizeEsp,originPoly,newPoly,objectiu)}
\]

\[
\begin{align*}
\text{numEsp} &= \text{generarRang}(1,5) \\
\text{Iterar tants cops com numEsp} \\
\text{Per cada vertex v de originPoly} \\
\text{Per cada vertex candidat c ja trobat } &\& !vInvalid \\
\text{vInvalid} &= (p >= c &\& p < c+sizeEsp/2-1) || \\
&\quad (c >= p &\& c < p + sizeEsp/2-1) \\
\text{Si vInvalid} \\
\text{Iterar al seguent v} \\
\text{Sino} \\
\text{posibleCandidat} &= \\
\text{CreaPoliliniaEsp(sizeEsp, originPoly,newPoly,v)} \\
\text{Si angle(posibleCandidat,objectiu) < angleCandidatActual} \\
\text{angleCandidatActual} &= \text{angle(posibleCandidat,objectiu)}
\end{align*}
\]
Si s’ha trobat candidat
resultat. AfegirCandidat
Retornar resultat

Pre: Rep dos polilínies, on la segona ha estat generada a partir d’extrudir la primera; juntament amb la mida i un vèrtex pertanyent a la primera polilínia
Post: Retorna la polilínia que representa l’espeleotema creat entre les dues polilínies, amb el vèrtex inicial i la mida seleccionats
CreaPoliliniaEsp(sizeEsp, originPoly, newPoly, v)
   Iterar entre sizeEsp/2-1 i 0
       resultat. afegirVertex(originPoly.obtenirVertex(v+i)
   Iterar entre 0 i sizeEsp/2
       resultat. afegirVertex(newPoly.obtenirVertex(v+i)
   Retornar resultat

Aquesta operació té un cost lineal en la mida de la polilínia d’extrusió. L’ordre que es segueix per crear la polilínia de l’espeleotema serà de forma que els vèrtexs estiguin en ordre horari i començant per la polilínia original, tal i com s’ha indicat a la funció CreaPolilinaEsp, la qual rep la mida de l’espeleotema, les dues polilínies de l’extrusió i el vèrtex candidat, i retorna la polilínia corresponent a l’inici de l’espeleotema.

3. Un cop obtinguts tots els candidats, caldría iterar per tots ells i crear l’espeleotema corresponent. En cas de què s’hagi de crear un pilar, es limitarà a crear només un per extrusió degut a què sinó visualment queda molt carregat, i sempre serà el primer vèrtex candidat. Després, es decidirà si tots els altres espeleotemes seran o bé estalagmites o bé estalactites mitjançant l’aleatorietat amb probabilitat 50 50.

• En cas de què l’espeleotema a crear sigui una estalagmita o una estalactita, primer de tot s’inicialitzaran els vèrtexs de la mateixa forma que per als túnels. Després, caldrà trobar el vèrtex simètric al baricentre de la polilínia respecte una línia horitzontal, per tal de trobar la distància màxima a la qual l’espeleotema pugui tenir sense intersecionar amb l’altra punta de la polilínia, com es pot veure a la figura 30. Aquest vèrtex simètric serà el que tingui menor diferència
Figura 28: Exemple de com es genera una estalactita a l’extrusió de a (verda) que genera la polilínia b (blava). En aquest cas, la polilínia inicial de la estalactita estaria formada per els vèrtexs $a_5,a_4,b_4,b_5$, que es correspondrien als vèrtexs de l’estalactita $e_0,e_1,e_2,e_2,e_3$ (en vermell)

D’angle entre el vector format pel vèrtex candidat i el vèrtex que s’està comprovant i el vector objectiu, que igual que abans, serà el vector up per les estalagmites i el vector down per les estalactites.

Un cop trobat aquest vèrtex, es calcularà la distància màxima de l’espeleotema, i es multiplicarà per un nombre aleatori corresponent al rang $[0.15,0.75]$, per tal de què a una mateixa extrusió els espeleotemes tinguin diferent mida i hi hagi d’aquesta forma més irregularitat.

A partir d’aquesta distància, es calcularà a quanta distància ha d’extrudir-se cada cop l’espeleotema i amb quin valor s’ha d’escalar, de la mateixa forma que es fa per a l’extrusió d’un túnel, ja que es tindrà com a constant la mida final de l’espeleotema (un valor molt proper a 0) i el número d’extrusions, un valor generat entre el rang 4 i 10. A partir d’aquests valors, ja es podrà extrudir de la mateixa forma que es fa amb un túnel, aplicant constantment l’operació d’escalat.

Pre: Rep dos polilínies, on la segona ha estat generada a partir d’extrudir la primera; juntament amb els altres paràmetres per generar l’espeleotema

Post: Crea un espeleotema a partir dels paràmetres seleccionats
Figura 29: Exemple de com es forma es troba el vèrtex simètric a partir d’un vector en la direcció de les y. En el cas del vèrtex a7, els dos vèrtexs amb l’angle més proper serien a1 i a2. En el cas de a4, no s’obtindria cap vèrtex al estar fora de la polilínia

CrearEstalagmitaEstalactita
  poliliniaCandidat.inicialitzar
  Per cada vertex de originPoly
    Obténir mínim angle(objectiu,vertexActual-\n    baricentreCandidat)
    mida = magnitud(baricentre, vertexSimetric)
    mida *= rang(0.15,0.75)
    numExtrusions = rang(4,10)
    distanciaExtrusio = mida/numExtrusions
    valorEscalat = Mathf.Pow (midaFinal / candidat.radi*2,\n                          1 / numExtrusions)
    Extrudir(poliliniaCandidat, distanciaExtrusio,valorEscalat)

• D’altra banda, si es tracta d’un pilar tindrem el punt des d’on començar a crear el que vindria a ser una estalactita, degut al vector objectiu que se li ha donat per trobar al candidat, el que vindria a ser l’inici del pilar o la part superior. Després caldrà trobar el final del pilar, el que vindria a ser l’inici de l’estalagmita corresponent a aquest pilar. És a dir, caldrà trobar el candidat amb una direcció d’extrusió més propera a up.
Caldrà fer una comprovació de què entre aquests dos vèrtexs inici i final del pilar hi hagi una direcció suficientment vertical per tal de què no es vegi com si estigués molt inclinat, ja que provocaria poc realisme.

A partir d’aquí es farà el mateix que per a les estalagmites i estalactites, extrudint amb una mida igual a la distància entre els punt inicial i final entre 2: es generarà una estalactita des de l’inici i una estalagmita des de el final. D’aquesta forma, al punt on acaba cada una que coincideix amb el punt mig, donarà l’impressió de què estan connectades i formen part del mateix objecte, com si es tractés d’un pilar.

Figura 30: Exemple de com seria un pilar sense limitar la direcció verticalment, quedant massa inclinat i evitant immersió

Cal comentar que inicialment la direcció en la qual s’extrudia era en la mateixa que la normal de la polilínia, però es va canviar a la direcció up o down segons si era estalagmita o estalactita per tal de què totes tinguessin la mateixa direcció i semblés més uniforme i natural.

Alguns dels problemes que es tenen a l’hora de generar aquests objectes, es que en extrusions on el radi de la polilínia és molt gran (és a dir, la polilínia és molt ample), però hi ha pocs vèrtexs, es veu un espeleotema massa gran degut que el mètode de creació agafa vèrtexs ja existents.
14.8 Triangulació

Tots els models d’objectes en gràfics no son més que un conjunt de triangles. Per tant, tota la generació de la cova gira al voltant de generar triangles, a partir dels vèrtexs de les polilínies. Abans de començar a explicar com serà el procés de triangulació cal tenir en compte dos aspectes importants:

- **L’Orientació de la cara** d’un triangle s’utilitza per pintar aquell triangle o no segons la posició de la càmera respecte la qual es dibuixa o renderitza l’escena. Si per un triangle la direcció d’aquesta orientació no està cap a la càmera, el triangle serà descartat i no es dibuixarà. Aquesta orientació és calculada segons l’ordre dels vèrtexs del triangle, depenent si és horari o antihorari, com es pot veure a la figura 31. Per tant, caldrà sempre que l’orientació dels triangles sigui cap a dins de la cova, ja que l’objectiu es explorar-la i per tant visualitzar-la des de dins.

- **El Sistema de coordenades** pot ser del tipus mà dreta o mà esquerra, segons com es calculi el producte vectorial de l’eix x i el y per obtenir el z, amb la regla de la mà dreta o esquerra respectivament. Cal determinar quin s’està utilitzant ja que la decisió de si un conjunt de vèrtexs està en sentit horari o antihorari depèn de quin sistema s’estigui utilitzant. **Unity** utilitza un sistema de mà esquerra, pel que es farà servir aquest. Per tant, per tal de pintar correctament un triangle, caldrà que els seus vèrtexs estiguin en ordre horari.

Així doncs, tenint en compte aquests aspectes, a continuació s’explica quan i de quina formarà es triangularà.

14.8.1 A l’Extrudir

Aquesta tasca va ser de les primeres en implementar-se i la qual no ha tingut masses canvis.

Degut a què en una extrusió hi ha el mateix número de vèrtexs per a les dues polilínies entre les quals s’ha de triangular i els vèrtexs estan units un a un de la forma en la que s’ha extrudit, es pot veure cada grup de quatre vèrtexs com un *quad*. És a dir, per cada vèrtex de la polilínia original, es formarà un *quad* amb el següent vèrtex adjacent en sentit horari i els dos vèrtexs corresponents a aquests de la polilínia creada a l’extrusió.

---

17Aquest procés s’anomena Culling
Figura 31: Exemple de culling. Degut a l’ordre dels vèrtexs, un sistema mà dreta i la direcció de la càmera (cap a la pantalla), el triangle de la dreta no es pintaria

Pre: Rep dos polilínies, on la segona ha estat generada a partir d’extrudir la primera
Post: Afegeix a la mesh tots els triangles de l’extrusió

$\text{TriangulaPolilinies}(p_1,p_2)$

Per a cada vèrtex de $p_1$
\[ v = \text{posició del vertex actual de } p_1 \]
\[ \text{TriangulaQuad}(p_1(v),p_1(v+1),p_2(v),p_2(v+1)) \]

Pre: Rep quatre vèrtexs, els dos primers corresponents a la polilínia original d’una extrusió, i els altres dos a la polilínia resultant de l’extrusió. Els quatre vèrtexs estan units formant un quad
Post: Afegeix a la mesh els dos triangles corresponents al quad

$\text{TriangulaQuad}(b_1,b_r,t_1,t_r)$

Si ni $b_1$ ni $b_r$ ni $t_1$ ni $t_r$ pertanyen a un forat
\[ \text{Generar triangle entre } b_1, t_r, t_1 \]
\[ \text{Generar triangle entre } b_1, b_r, t_r \]

Cada $\text{quad}$ generarà dos triangles, degut a què dos triangles formen un quadrat. Per tant, per a cada extrusió, es generarà $2n$ triangles, sent $n$ el número de vèrtexs que forma una polilínia.
Figura 32: Triangulació de l’extrusió de la polilínia \( b \) (blau) a partir de \( a \) (verd), estant els triangles marcats en vermell. Com a exemple i seguint el pseudocodi, es segueix la nomenclatura \( bl, br, tl, tr \) per indicar a cada iteració el vèrtex inferior esquerra, inferior dret, superior esquerra i superior dret respectivament del quad.

### 14.8.2 Al fer forats

Al principi es triangulava un forat com una extrusió més, ja que a partir dels vèrtexs marcats per a fer un forat, es creava una polilínia i era tractada com l’inici d’un túnel. Això, com es comenta a l’apartat 14.6, provocava a la llarga formes estranyes a la cova, i per tal de solucionar-lo, a partir d’aquesta polilínia es generà una altra utilitzant mètodes de projecció. Aquesta nova polilínia va passar a ser tractada com a l’inici del túnel creat amb el forat, per tal d’evitar el problema anomenat i poder tenir així més control de com es fa la generació. Així doncs, es va haver d’idear una manera per triangular entre aquesta polilínia formada a partir d’alguns dels vèrtexs de les dues polilínies d’una extrusió i entre la nova polilínia que vindria a ser l’entrada del nou túnel.

Aquesta triangulació es fa a partir de la idea del *merge* de l’algorisme de *Divide and conquer* per a trobar envolupants convexes d’un conjunt de punts en 3D[26]. Es tracten les dues polilínies entre les quals es vol triangular com si fossin les dues envolvents convexes que es volen fusionar per a trobar la envolupant convexa que formen entre les dues. D’aquesta forma, es troben tots els triangles que uneixen les dues polilínies.
L'algoritme comença amb un vèrtex de cada polilínia, anomenem-los a i b. S'agafarà el següent vèrtex de cada polilínia en sentit horari, a’ i b’, i donat que a la polilínia ja estan guardats seguint aquest ordre aquesta operació té un cost constant. A continuació, si l’angle format pels segments ab i aa’ és menor que el format pel ba i bb’, s’agafarà com a guanyadora la primera polilínia, es triangularà entre aa’b, i a passarà a valdre a’. Intercanviant els rols de a i b, s’obté anàlogament el cas on la segona polilínia és la guanyadora.

En aquest cas a cada iteració només es té un candidat per polilínia, mentre que a l’algoritme original on es té qualsevol envolupant convexa, és molt estrany que es produeixi aquest fenomen. Al trobar-se en unes condicions diferents que a l’algoritme general, es va haver d’afegir un límit de quants cops un mateix vèrtex podia ser escollit com a guanyador ja que degut a què la polilínia que representa el forat no sempre és convexa, es podien arribar a crear alguns artifacts. Aquest límit pren un valor molt petit.

Tot i així, hi ha casos extrems on el resultat no és gaire bo, com es pot apreciar a la figura 33.

**Pre:** Rep dos polilínies, la primera corresponent a un forat creat a partir d’una extrusió, i la segona la projecció de la primera polilínia

**Post:** Triangula entre les dues polilínies

```
TriangulaForat(p1, p2)
```

```
a = b = 0
Mentres a < numvertexs(p1) && b < numvertexs(p2)
    a' = a+1
    b' = b+1
    aAngle = angle(p1(a)p2(b),p1(a)p1(a'))
    bAngle = angle(p2(b)p1(a),p2(b)p2(b'))
    Si a no ha canviat durant maxWin
        guanyador = b
    Altrament si b no ha canviat durant maxWins
        guanyador = a
    Altrament
        Si aAngle < bAngle
            guanyador = a
        Altrament guanyador = b
```
Si guanyador == a
    Generar triangle p1(a),p1(a’),p2(b)
    a = a'
Altrament guanyador == b
    Generar triangle p1(a),p2(b’),p2(b)
    b = b'
Si a >= numvertexs(p1)
    Mentres b < numvertexs(p2)
        Generar triangle entre p1(a), p2(b), p2(b+1)
        b++
Altrament b >= numvertexs(p2)
    Mentres a < numvertexs(p1)
        Generar triangle entre p2(b), p1(a), p1(a+1)
        a++

Figura 33: Es pot observar com el forat no es triangula equitativament, produint tancaments. Si no es fiqués el límit de iteracions seguides on un vèrtex pot ser guanyador, aquesta situació seria més freqüent.

El número de vèrtexs generats per tant serà $\theta(m+n)$, on $m$ és el numero de vèrtexs agafats de l’extrusió per generar el forat i $n$ el número de vèrtexs de la polilínia que forma l’inici del nou túnel. $n$ serà sempre major que $m$, pel que el cost es redueix a $\theta(n)$.

14.8.3 Al tancar un túnel

Un cop s’ha finalitzat la creació d’un túnel, bé perquè s’ha extrudit fins al màxim de cops permesos o bé perquè s’ha detectat una intersecció al haver-hi un tros
de la cova generada prèviament, cal tancar el túnel actual.

Per tal de tancar el túnel, caldrà trobar un punt mig amb el qual tots els vèrtexs de la polilínia s’uniran i produiran el tancament. Això és simplement trobar el baricentre de la polilínia fent la mitjana de tots els vèrtexs, i finalment triangular els vèrtexs amb aquest baricentre. La triangulació serà molt senzilla: constarà en un bucle que recorrerà tots els vèrtexs i formarà un triangle amb el vèrtex actual, el següent i el baricentre. S’ha escollit aquest ordre degut al que s’ha comentat sobre l’orientació de la cara i el sistema de coordenades.

Pre: Rep una polilínia corresponent a la última d’un túnel.
Post: Afegeix a la mesh el tancament de la polilínia paràmetre triangulant amb el baricentre d’aquesta

Tancar polilínia(p)
   b = baricentre(p)
   Per a cada vertex de p
      v = posicio del vertex actual de p
      Generar triangle entre p(v),p(v+1),b

Per tant, el número de triangles generats són \( \theta(n) \), on n és el número de vèrtexs de la polilínia.

14.8.4 Espeleotema

Degut a què els espeleotemes es creen de la mateixa forma que un túnel tal i com s’ha explicat al seu apartat corresponent, el número de triangles generats són els mateixos: \( \theta(n \cdot e + n) \). n és el número de vèrtexs que formen una polilínia de l’espeleotema i e és el número d’extrusions que es fan per a un espeleotema, on n sempre serà uns valor molt petit, i e tindrà un valor constant i també petit per raons de construcció de l’espeleotema.

La suma es deu a què cal afegir el cost de tancar un túnel, que vindria a ser un espeleotema en aquest cas.

En el cas dels pilars, aquest valor es doblarà degut a què s’està generant una estalagmita i estalactità a l’hora. En qualsevol cas, el cost final de generar un espeleotema acaba sent \( \theta(n \cdot e) \).

Degut a què a diferència de la cova, es vol que els triangles dels espeleotemes siguin vists des de fora, és a dir, no com si estiguéssim a dins de l’espeleotema;
caldrà fer una excepció i canviar l’ordre dels vèrtexs per a generar els triangles i que siguin visibles des de fora.

14.8.5 Cost de la triangulació

Degut a què el cost de crear i afegir un triangle a la mesh és constant i que la resta d’operacions de la triangulació no afegeixen cap cost addicional, es pot veure que el cost total de la triangulació és igual als triangles generats. A més, aquest cost és sempre lineal segons el número de vèrtexs de la polilínia.

Els paràmetres maxExtrudeTimes i maxHoles representen el número d’extrusions màximes per túnel i de túnels màxims respectivament. Degut a les possibles interseccions, aquest valors no seran sempre exactes, sinó que seran el valor màxim que es pot arribar a tenir. Pel càlcul dels triangles totals generats als forats, es farà servir el paràmetre holesMaxVertices, que indica el número màxim de vèrtexs que es poden utilitzar per generar un forat; que de la mateixa forma no sempre s’arribarà a aquest límit.

El número de espeleotemes creats sempre serà inferior al número total d’extrusions, pel que es pot utilitzar aquest valor com a fita.

Com s’ha vist, el cost de generar un espeleotema és lineal en n, pel que s’utilitzarà aquest cost.

Per tant, el número de triangles totals que es generaran a una execució, així com el cost de crear-los sense tenir en compte els costos de les interseccions i el suavitzat, serà de l’ordre de $O(2 \cdot n \cdot \text{maxExtrudeTimes} \cdot \text{maxHoles} + (\text{HolesMaxVertices} + n) \cdot \text{maxHoles} + n \cdot \text{maxHoles}) + n \cdot e \cdot \text{maxExtrudeTimes} \cdot \text{HolesMaxVertices}$, el que vendria a ser $O(n \cdot \text{maxExtrudeTimes} \cdot \text{maxHoles})$.

Es pot veure que no és massa complexe per tant el cost de la generació. S’ha demostrat en apartats anteriors que el cas pitjor de generar qualsevol operació és sempre lineal respecte el nombre de vèrtexs de polilínia. Per tant, es justifica el cost total fins al moment de $O(n \cdot \text{maxExtrudeTimes} \cdot \text{maxHoles})$.

Finalment, faltaria tenir en compte la comprovació de la intersecció, el qual, tal i com es demostra al següent apartat, és el coll d’ampolla ja que es passa a tenir un cost quadràtic en un hipotètic cas pitjor que absolutament mai s’assolirà. Per aquesta única raó de les interseccions, no es podria agafar el cost
anterior com a cost total de la generació final.
Altres aspectes com la texturització o el suavitzat no afegeixen cost addicional tal i com s’explica als corresponents apartats.

14.9 Mesh

S’utilitza una estructura de dades pròpia per representar la mesh, que més tard es transforma a una mesh de Unity.

Aquesta estructura de dades contindrà la següent informació:

- Llista de posicions dels vèrtexs
- Llista de coordenades de textura dels vèrtexs. Juntament amb l’anterior, quan s’afegeix una nova polilínia s’afegeix tant la posició a l’espai com la de la textura de tots els vèrtexs, en ordre de vèrtex. Per tant, han de tenir la mateixa mida
- Llista de triangles: Al acabar una extrusió, crear un forat, un espeleotema o bé tancar un túnel, s’afegeiran tots els triangles generats. Ha de ser múltiple de 3 degut a què per cada triangles es té els índexs dels vèrtexs pels quals està format. Per cada triangle, els vèrtexs han de mantenir l’ordre anomenat a l’apartat 14.8. Sempre que es vagi a generar un nou vèrtex, per saber l’índex el qual donar-li s’utilitzarà el numero de vèrtexs creats fins al moment.
- Set amb la llista de vèrtexs que pertanyen a un forat, per tal de poder marcar-los per més tard poder aplicar un suavitzat només a aquests vèrtexs. A mesura que es vagi creant forats, s’afeigrà els índexs dels vèrtexs corresponents.

Inicialment s’utilitzava només una mesh pròpia que era transformada directament a una de Unity per tal de guardar tota la informació sobre la triangulació i el renderitzat. A mesura que s’anaven implementant millores, especialment la de generar més vèrtexs per tal de tenir una forma més suavitzada a la cova i més agradable visualment, es va trobar amb el problema de què es sobrepassava el límit de vèrtexs per mesh permesos per Unity, el qual és aproximadament 65536 degut a què per a guardar els índexs s’utilitzen enters de 16 bits. Es van considerar les següents solucions:
• El primer que es va pensar per resoldre això va ser en un cop acabada la generació, dividir la pròpia mesh en diferents submeshes, el que vindria a ser realment tenir diverses meshes. El principal problema d'aquest mètode era que no es sabia molt bé quan ni com dividir la mesh, ja que no es podia fer en l'ordre en el qual anaven apareixen els diferents vèrtexs degut a què per exemple, en la versió recursiva aquest no estarien ordenats per l'ordre en el que s'anaven triangulant.

• L'altra alternativa i la que es va aplicar és dividir les meshes a mesura que s'anava fent la generació. Per resoldre el problema de quan s'havia de fer el canvi de mesh, es va decidir una solució molt intuitiva: cada túnel seria representat per una mesh diferent.

Per als espeleotemes, s'utilitza una mesh a part, per tal de tenir-ho separat i evitar possibles problemes d'indexació de vèrtexs per a triangular. Si es veu que s' està a punt d'arribar al número màxim de vèrtexs permesos per mesh, es genera una nova mesh per evitar aquest problema.

Tal i com es generen els índexs quan es crea un nou vèrtexs, es té un cost constant a l'hora d'afegir un nou vèrtex i triangle, juntament amb l'estructura de dades utilitzada[27].

És molt senzill transformar la pròpia mesh a la que utilitza Unity, ja que només cal passar-li la direcció de memòria de les llistes amb la posició dels vèrtexs, les coordenades de textures i els índexs per als triangles, pel que té cost constant. Tot i així, al no tenir informació sobre les normals, cal calcular-les amb una funció pròpia de Unity, la qual recorre tots els triangles i es guarda la normal dels triangles adjacents de cada vèrtex. Un cop s'ha obtingut això, es fa la mitjana d'aquestes normals. Això té un cost \( \theta(e \cdot \log n + n) \), on \( e \) és el nombre de triangles i \( n \) el nombre de vèrtexs totals. Equival a \( \theta(e \cdot \log n) \) degut a què sempre es complirà que \( e > n \).

Una alternativa a fer-ho així, hauria sigut mantenir una estructura de dades on cada vèrtex té una referència als triangles adjacents, de forma que a mesura que es crea un triangle, s'afegiria a aquesta estructura. Això hagués requerit l'ús d'un Hash, tenint un cos logarítmic el afegir un triangle en comptes de constant, només per estalviar-nos un cost \( \theta(e \cdot \log n) \) a l'acabar la generació, ja
que utilitzant aquesta estructura només caldria recórrer tots els vèrtexs un cop per adaptar-los a la forma en que Unity ho necessita. Per tant en aquest cas s’està perdent eficiència. Tot i així, aquesta estructura segurament seria la més adient si es volgué modificar vèrtexs i triangles un cop han estat creats, com és el cas de les millores que es proposen com a treball futur, ja que necessiten accedir constantment als triangles i vèrtexs generats.

Guardant les normals d’aquesta forma, però, s’evitaria un dels problemes que es té actualment: al haver-hi diverses meshes i existir vèrtexs replicats, la normal que es té per aquests vèrtexs corresponents a les polilínies d’inici de túnel és diferent per a la mesh del túnel que ha creat aquesta polilínia com a forat, i per a la mesh del nou túnel. Això es degut a què tenen diferents triangles adjacents, i per tant la normal té un valor diferent i es nota per la il·luminació, en algunes textures més que en altres. Aquest problema també es té per als espeleotemes, al pertànyer també a una mesh diferent que la del túnel on s’ha generat.

Figura 34: Es pot observar el canvi de la il·luminació en els vèrtexs replicats, tant per forats com per espeleotemes. Aquest efecte s’accentua en textures amb colors molts uniformes, com en aquesta textura de neu

14.10 Interseccions

Per tal de evitarr les possibles interseccions entre les diferents parts de la cova, es van plantejar tres mètodes:

- Utilitzar Bounding Boxes. Cada cop que es generi un tros de la cova, es creàrà una Bounding Box d’aquest tros. D’aquesta forma, cada cop que es vagi a fer una extrusió, caldrà comprovar si aquesta intersecciona amb totes les altres Bounding Boxes existents fins al moment. Per tal de reduir el número de capses totals, s’aniran acumulant totes les polilínies
d’extrusions consecutives fins fer un canvi de direcció, o bé crear un forat en el cas del mètode recursiu. Quan succeixi qualsevol d’aquestes dues opcions, es generarà una nova Bounding Box amb les polilínees acumulades i es tornarà a començar el procés d’acumular-les per generar la nova capsà. El problema d’aquest mètode seria el cas de què s’anessin produint molts girs, que incrementaria el número de capses. Existeixen dos tipus diferents de Bounding Boxes:

- **Axis Aligned Bounding Box (AABB)**, faciliten molt la seva creació i la comprovació d’intersecció entre dues Bounding Box d’aquest tipus. El problema que tenen es que es poden generar capses molt grans al haver-hi un tros de cova amb una direcció d’extrusió molt inclinada, com es pot observar a la figura

![Figura 35: Ocupació a l’espai de les AABB. Es pot observar com es desaprofita molt espai que realment està lliure](image)

- **Object Oriented Bounding Box (OOBB)**, són mes complexes de crear però permeten més precisió en quan a l’espai real ocupat per cada tros. També tenen la desaventatge que és més costós comprovar la seva intersecció, així com al seva creació.

- Fer una subdivisió de l’espai mitjançant **Octrees**, de forma que cada cop que es faci una extrusió, es marcarà com a visitat el subespai que ocupi. D’aquesta forma, cada cop que es vulgui fer una extrusió, caldrà comprovar que el subespai corresponent no estigui prèviament visitat. El principal problema d’aquest mètode seria que en cada node del octree podria haver-hi més d’una polilínia, o bé que una polilínia estigués en més d’un node a la vegada.
Finalment s’ha decidit implementar el mètode de AABB degut a ser el més equilibrat en simplicitat i rendiment, així com donar uns resultats bastant acceptables. Respecte el cost computacional, tot i ser pitjor que la tècnica d’octrees, com que el número de capses generades no serà massa elevat, no té una repercussió massa negativa en el rendiment. Tot i així, és el coll d’ampolla del cost computacional.

Per tal de generar una nova capsa, s’aniran acumulant les polilínies que es van generant durant les extrusions, fins arribar a algun dels dos casos següents:

- Generar nova direcció, degut a utilitzar AABB si no es fes d’aquesta forma es tindria molt ocupació de l’espai per un tros de túnel que realment estaria molt desaproveitat, tal i com es pot veure a la figura 35. No és imperatiu per evitar les interseccions i generar més capses de les necessàries, però permet més llibertat i espai a l’hora d’extrudir.

- Al crear un forat en la versió recursiva, ja que no es vol que el nou túnel creat interseccioni amb el túnel actual, pel que realment cal crear les caixes en aquest punt.

Per tant, quan s’arrivan a un d’aquests casos es recorreran totes les polilínies i es formarà la AABB que formen entre elles. Això té un cost $\theta(n \cdot m)$, on $n$ és el número de vèrtexs per polilínia i $m$ el número de polilínies acumulades. De totes formes, com que cada polilínia només serà afegida un cop a una AABB, el cost total de generar totes les capses és de $\theta(n \cdot \text{maxExtrudeTimes} \cdot \text{maxHoles})$. Com es pot observar el coll d’ampolla de la generació no prové de generar les capses sinó d’estar constament comprobar les interseccions.

Les interseccions es comprovaran cada cop que es generi una extrusió o un forat, per comprovar si realment poden ser creats sense problemes. Aquesta comprovació es farà com indica al següent pseudocodi:

```
Pre: Rep la bounding box formada per dues polilínies d’una extrusió, així com el paràmetre el qual indica l’índex de la capsa amb la qual es pot interseccionar
Post: Retorna si existeix una intersecció entre la capsa seleccionada i les creades fins al moment
Interseccio(extrusio, excepcio)
```
per a totes les capses creades
   si capsaActual != excepcio \&\& intersecció(extrusio, capsaActual)
       Retorna true
   Retorna false

A mesura que es van generant les capses es van afegint a la llista de AABB, de manera que cadascuna té un índex que la representa igual a la posició a aquesta llista.
D’aquesta forma, el paràmetre excepció indica l’índex d’una capsa amb la qual es pot interseccionar. Això té sentit en diverses ocasions:

- Al començar a generar un nou túnel després de fer un forat a una crida recursiva. L’índex és el de la capsa que s’ha creat just abans de generar el túnel. A la que es genera una nova capsa al nou túnel a causa d’un canvi de direCCIó o creació de forat es canvia aquest índex a -1, de forma que qualsevol capsa provoca una interseCCIó.
Es posa una excepció en aquesta condició per tal de què el nou túnel pugui tenir més llibertat a l’hora d’extrudir-se degut a les AABB, tot i que en ocasions si aquesta és molt gran fa que a la que es posa l’índex a -1 ja no es pugui extrudir més degut a què es troba interseccionant amb la capsa que abans s’estava ignorant. Aquesta capsa no es pot mantenir constant per tot el túnel ja que degut als canvis de direCCIó podria provocar una interseCCIó.

- Al tornar de la crida recursiva, amb la primera capsa generada pel nou túnel. Degut a què els canvis de direCCIó tenen un límit que mai sobrepassarà els 180°, ficar com a excepció aquesta capsa no produirà cap interseCCIó. S’evita el mirar la interseCCIó d’aquesta capsa per tal de què el túnel actual tingui més llibertat a l’hora d’extrudir-se.

- Al obtenir el següent forat a qualsevol versió iterativa. Juntament amb el forat i per tant l’inici del túnel, s’obté l’índex de la capsa, la qual serà la següent capsa creada per el túnel actual just després de fer el forat, per les mateixes raons que a la versió recursiva.

- Al fer un canvi de direCCIó i generar una nova capsa. Es posarà aquesta capsa com a excepció per la mateixa raó que per quan es posa al començar a extrudir un nou túnel.
Per tant, al cost total de la generació obtingut a l’apartat 14.8.5, $O(n \cdot \text{maxExtudeTimes} \cdot \text{maxHoles})$, caldria multiplicar-ho pel cost de comprovar si hi ha o no interseccions a cada extrusió, que en el cas pitjor i posant una fita molt exagerada i poc acotada, seria el mateix cost que número d’extrusions total, és a dir, $O(\text{maxExtrudeTimes} \cdot \text{maxHoles})$.

És pot concloure d’aquesta forma amb una fita del cost total de l’algoritme, que seria $O(n (\text{maxExtudeTimes} \cdot \text{maxHoles})^2)$, tot i que per tal d’arribar a aquest cas pitjor, caldria que s’estés generant una capsà a cada extrusió. Això equivaldria a estar generant un forat o bé fent un canvi de direcció a cada extrusió, i aquesta és una situació a la qual mai s’arribarà.

14.11 Smooth

Per tal de tenir una millor visualització de la cova, s’ha utilitzat un suavitzat dels triangles, quedant així més arrodonida i suavitzada.

S’han estudiat diversos mètodes per tal de fer aquest suavitzat, i es podrien dividir en dos grans subgrups: Els que apliquen el suavitzat just al fer una extrusió i els que apliquen el suavitzat un cop ja s’ha generat tota la cova.

14.11.1 Extrusió

Aquests mètodes pretenen, a partir de una polilínia existent, crear-ne una de nova amb més vèrtexs de forma que tingui més resolució i per tant ajudar a donar un efecte de suavització.

Existeixen molts algoritmes per tal de dur a terme aquesta subdivisió de la polilínia, a continuació s’esmenten els que s’han considerat:

- B-Spline. A partir d’un cert número de vèrtexs, es crea l’equació d’una corba entre els punts. Aquesta corba no es més que un polinomi de grau parametrizable. A partir d’aquesta equació, es poden obtenir els punts desitjats que pertanyin a aquesta[28].

- Corbes de Bézier, és un tipus de B-Spline amb més limitacions, degut a què sempre utilitza el polinomi de Bernstein[29].

- Mitjana de veïns. És un mètode molt simple, però que funciona força bé per obtenir un efecte de suavitzat.

Per a cada parell de vèrtexs adjacents, crea un nou vèrtex amb una posició igual a la mitjana del parell. Si el suavitzat finalitzés aquí, realment
no s’obtindria cap resultat diferent, només un increment en el número de vèrtexs. El que es fa per tal d’obtenir un suavitzat és, utilitzar els nous vèrtexs per obtenir un nou valor dels vèrtexs antics a través de la mitjana dels seus dos nous vèrtexs adjacents.

Figura 36: Exemple de la subdivisió de vèrtexs utilitzant la mitjana de veïns durant tres iteracions: en cada iteració es dobla el número de vèrtexs i es pot observar com la polilinia es va fent petita, especialment al principi, sent aquesta una de les conseqüències de guanyar resolució. La polilinia final es la dibuixada com a negre, i a cada iteració es veu com es fa la mitjana dels vèrtexs obtinguts a la iteració anterior.

Per les propietats de la polilinia, cada vèrtex només té dos adjacents i estan ordenats, pel que el cost d’aquest suavitzat és lineal respecte el número de vèrtexs de la polilinia.
A cada crida d’aquest mètode es duplica el número de vèrtexs, pel que el número final de vèrtexs que formen la polilïnia és exponencials respecte al número de cops que s’aplica el suavitzat.

14.11.2 Generació total

Existeixen un altre cop, diverses mètodes. Tots són mètodes generalitzats per al suavitzat de malles arbitràries. Es van considerar els següents a partir d’observar les opcions smooth i subdivide smooth de Blender:

• Filtre laplaciana: És un mètode utilitzat normalment per malles amb soroll. Bàsicament, un filtre laplaciana el que fa es fer servir mitjanes per tal de filtrar aquest soroll i suavitzar així la malla. Per cada vèrtex, s’utilitza la mitjana de la posició dels vèrtexs adjacents com a nova posició[30, 31], repetint l’operació durant un número d’iteracions concretes. Existeixen altres mètodes més avançats que utilitzen aquesta tècnica, però s’ometran per la seva complexitat[32].

• Subdivisió Catmull-Clark: Fa una subdivisió recursiva de la malla de forma que es creen nous vèrtexs i triangles fent servir mitjanes, de forma que com més crides recursives es facin, més suavitzat però també més triangles es crearan. Aquest mètode es pot veure com si s’estigués afegint resolució a cada triangle[33]. Aquest mètode és utilitzat no només per Blender, sinó també per molts altres softwares de modelatge.

14.11.3 Mètode final

Finalment, s’ha fet servir una barreja dels dos mètodes dividits: Cada cop que s’inicia un nou túnel, s’aplica un suavitzat d’extrusió a la polilïnia inicial, de forma que a cada extrusió s’estarà generant una nova polilïnia ja suavitzada. Es fa servir el mètode de mitjana de veïns, fent servir el valor parametrizable smoothItIniTunel com a número de cops que s’aplica el suavitzat. Per tant, el número de vèrtexs de la polilïnia després d’aplicar el suavitzat, serà \( \theta(i \cdot 2^S) \), on i serà el número de vèrtexs abans del suavitzat i S les iteracions a aplicar, sent aquest valors els paràmètres gateSize i smoothItIniTunel respectivament. Degut a què és un cost exponencial, cal triar un valor de S molt petit. Per exemple amb i = 4 i S = 4 són amb els que s’han fet totes les captures, i s’obtenen resultats bastant bons com es pot apreciar.
Addicionalment, degut a què la distància per una extrusió no serà gaire gran, no es tindrà un canvi molt brusc entre dos polilínies consecutives, independentment de l’operació d’extrusió que s’estigui aplicant. Això es com fer un suavitzat implícit entre extrusions: es com si entre dues extrusions amb molta distància es fessin diverses extrusions petites que combinades, formen l’extrusió gran, però amb més resolució, degut a què s’ha incrementat el número de triangles. D’aquesta forma, per algunes de les operacions d’extrusió es dividirà el valor que s’aplica:

- **Rotació**: Simplement a partir del valor generat aleatoriament, es divideix entre el número d’extrusions que s’aplicarà l’operació de rotació de forma que a cada extrusió s’aplica el mateix valor. Per tant, la suma de tots els petits valors de rotació és igual al valor de rotació generat.

- **Canvi de direcció**: A partir de la direcció generada, a cada extrusió s’anirà aplicant una interpolació entre la direcció d’extrusió actual i la generada. D’aquesta forma, el valor d’interpolació el primer cop que s’apliqui l’operació valdrà 0 per ser igual que la direcció actual, i s’anirà incrementant equitativament fins arribar a l’última extrusió on s’aplica l’operació que valdrà 1, i per tant serà igual a la direcció generada. Fer servir aquesta interpolació ajuda molt a crear un efecte de suavitzat.

- **Escalat**: Degut a què l’escalat sempre es fa a partir de la mida actual de la polilínia, no es pot dividir el valor generat entre el número d’extrusions que dura l’operació, ja que acabarien obtenint-se resultats diferents allí que s’espera. Això es degut a què l’escalat realment el que fa és multiplicar la mida, pel que és una funció exponencial en aquest cas al estar-se fent extrusió darrere extrusió. Per tant, a cada extrusió que duri l’operació se li haurà d’aplicar un escalat igual a l’arrel i-èssima del valor generat, on i és la duració de l’operació.

Els mètodes de generació final s’han fet servir per suavitzar els forats. Un forat, degut a què es crea a partir d’una extrusió, sempre tindrà els seus vèrtexs dividits entre els de la polilínia extrudida i la polilínia obtinguda de l’extrusió, de forma que estan unides per l’aresta que es forma amb la pròpia extrusió dels vèrtexs que marquen el límit del forat, tal i com es pot veure a la figura 26. A més, estan units tots aquest vèrtexs que formen el forat amb el nou túnel, mostrant un resultat molt poligonal sempre. Per tant, cada cop que es genera un forat, es marquen els vèrtexs corresponents a aquest, i s’aplica el
smooth Laplacíà a aquests un cop generada la cova per a tal de mostrar un millor resultat. El número d’iteracions que es faran servir estarà marcat per el paràmetre smoothItAfterGeneration.

Quan s’apliqui aquest suavitzat primer de tot s’obtindran els vèrtexs adjacents dels vèrtexs marcats: això té un cost $O(e \cdot \log h)$, on $e$ és el número de triangles totals i $h$ el número de vèrtexs marcats, sent aquest cost el mínim per mantenir l’estructura de dades on es guarda la informació dels adjacents. Un cop trobats els adjacents, caldrà recórrer tots els adjacents de cada vèrtex a cada iteració per tal de trobar la mitjana, tenint així un cost $\theta(h)$. És un cost constant en el número de vèrtexs marcats degut a la propietat de què entre una mateixa polilínia un vèrtex només tindrà dos adjacents, i entre dues polilínies només es tindrà el vèrtexs adjacent de l’extrusió.

Degut a què això només té en compte els forats, es podria agafar $h$ com $O(\text{maxHoles} \cdot n)$, amb $n$ igual al número de vèrtexs per polilínia. per tant, aquesta operació de suavitzat en conjunt no afegeix cap cost addicional al algoritme.

![Imagen](image1.png)  ![Imagen](image2.png)

Figura 37: Es pot observar com al haver-hi un forat al final del túnel, si no es duplica la polilínia es creen triangles massa exagerats

D’aquest procés va derivar un problema: quan es tancava un túnel just després de fer un forat, el suavitzat modificava la posició d’alguns vèrtexs que estaven connectats amb el baricentre que permetia el tancament i provoca resultats com els que es poden observar a la figura 38.

Per tal de solucionar-ho, quan es produeix aquest cas es duplica l’última polilínia del túnel, és a dir, la que està connectada al forat i al tancament del túnel, i s’uneix i es triangula amb l’original, com si es tractés d’una extrusió. D’aquesta forma, el suavitzat s’aplica només a la polilínia original i gràcies a la còpia s’evita el problema anomenat.
14.12 Texturització

Tot i que la texturització era un dels objectius secundaris, aporta molta qualitat a la visualització de la cova generada degut a què és l’únic que li aporta color. S’han obtingut uns resultats molt millor dels esperats, però hi ha petits problemes a resoldre.

14.12.1 Extrusió

Per tal de tenir una texturització visualment agradable, cal que hi hagi continuïtat entre les diferents extrusions. Per això, es farà servir textures circulars obtingudes prèviament. Això vol dir que, degut a què són textures en 2D i per tant imatges, la part dreta de la imatge caldrà que sigui continu a amb la part esquerra, i el mateix amb la part superior i inferior, com si es tractés de tiles.

![Figura 38: Exemples de textures circulars. Es pot observar la continuïtat de la imatge en els costats](image)

Per a assignar les coordenades de textura, es farà un *UV mapping* de la textura a l’extrusió, de forma que es com si s’emboliquis l’extrusió feta amb la imatge que representa la textura, canviant la mida d’aquesta per a què s’adapti a l’extrusió.

D’aquesta forma, per a una polilínia se li assignarà una coordenada de textura $x = 0$ al primer vèrtex, i $x = 1$ a l’últim, de forma que entre tota la polilínia es cobreixi la coordenada $x$ de la textura\(^{18}\). Això va tenir com a conseqüència

\(^{18}\)Les coordenades de textura sempre pertanyen al rang $[0,1]$, utilitzant en aquest cas la
que el primer vèrtex es va haver de duplicar, amb la mateixa posició a l’espai, però amb diferent coordenada de textura, tenint una x que val 0 i 1 respectivament. D’altra forma, es provocava que amb el que era abans últim vèrtex de la polilínia i el primer, es texturés de forma estranya ja que es feia entre un valor molt proper a 1 i 0, en comptes de entre aquest valor proper a 1 i 1, formant així una espècie de textura comprimida.

Respecte a la coordenada y, comença amb 0 per a l’inici de la cova, i es va

![Figura 39: Si no es dupliqués el primer vèrtex per tal de què tingui dos coordenades de textura, es veuria contínuament aquesta espècie de línia que no és més que la textura completa en un espai molt petit, produint aquest resultat incrementant a cada extrusió amb un valor proporcional a la distància d’extrusió. D’aquesta forma, entre dues polilínies d’una extrusió es com si es formés un rectangle a la textura, ocupant tot l’espai en x i un tros de l’espai en y. Així doncs, els vèrtexs d’una mateixa polilínia tindran tots la mateixa coordenada de textura y, però diferent x, la qual pertanyerà sempre al rang [0,1] i estarà en ordre ascendent segons els vèrtexs.

A continuació es pot observar el procediment que es segueix per assignar les coordenades x a la polilínia inicial d’un túnel:

**Pre:** Rep la polilínia inicial d’un túnel

**Post:** Genera les coordenades de textura de la polilínia inicial

**GenerarUVS(p)**

```plaintext
distanciaP = 0
Per cada vertex v de p
distanciaP += distancia(p(v),p(v+1))
p(0).UV = (0,0)
```

opcio de repetició de la textura si estan fora d’aquest rang, és a dir, prendre el valor decimal
Figura 40: Texturització de l’extrusió de la polilínia a (verd), que genera b (blau). Es pot veure com la triangulació (vermell) rep unes coordenades de textura que van amb una x entre 0 i 1 proporcional a la distància entre vèrtexs.

Per cada vertex v de p excepte el primer

\[ \text{distAux} = \text{distancia}(p(v-1), p(v)) \]

\[ p(v).UV = p(v-1).UV + \left( \frac{\text{distAux}}{\text{distanciaP}}, 0 \right) \]

Per tant, quan s’està extrudint una polilínia, el vèrtex que es genera a partir d’un altre té una valor de coordenada de textura igual a l’original més un increment en y, mantenint la mateixa x. D’aquesta forma la textura s’anirà repetint en y a mesura que es vagi fent extrusions.

La forma en que s’extrudeix una polilínia centrant-nos només amb les coordenades de textura seria la següent, on \( UV\text{factor} \) és la constant de proporcionalitat:

Pre: Part del codi de l’extrusió centrat en les coordenades de textura, per tant es rep la polilínia a partir de la qual extrudir
Post: Genera les coordenades de textura de la polilínia creada a l’extrusió

\[ \text{Distancia} = \text{obtenirDistanciadeOperacioActual} \]
\[ \text{IncrementUV} = (0, \text{Distancia} / UV\text{factor}) \]

Per cada vertex

Extrudir nou vèrtex a partir del vertex actual

\[ \text{CoordenadesUVNouVertex} = \text{CoordenadesVertexOriginal} + \text{IncrementUV} \]

Cal comentar que inicialment es va plantejar utilitzar la direcció \((x,z)\) en la qual s’extrudia la polilínia com a increment de textura, però hi havia problemes quan un túnel tenia una direcció d’extrusió molt al límit, juntament amb els
problemes que s’afegeixen al fer un forat.

14.12.2 Forat

El principal problema que es té per a la texturització al seguir un mapeig simple a les extrusions és al fer un forat, ja que s’ha d’interpolar entre un conjunt de vèrtexs de polilínies diferents (i per tant formant un rectangle a la textura) amb una nova polilínia. Aquest problema es deriva de què no es pot interpolar correctament de cap forma entre aquestes dues figures, ja que sempre s’haurà d’interpolar entre dues posicions molt extremes, amb el que s’obtenen resultats no gaire agradables a la vista.

Figura 41: Es pot veure com per a la polilínia forat (blau) formada a partir d’alguns vèrtexs de dos polilínies (a i b) d’una extrusió, és complicat que s’interpoli correctament amb la polilínia projecció (p, verd), degut a què la distància entre vèrtexs de cada polilínia no serà proporcional, a més de no tenir el mateix número de vèrtexs. És a dir, no es podrà interpolar correctament entre la polilínia \( p_0, p_n \) i la polilínia \( a_i, a_j, b_j, b_i \).

Aquest problema es podria considerar com el que repercuteix més negativament a tot el projecte, ja que és el que prova una pitjor visualització en ocasions al canviar de túnel.

Es va plantejar fer un mapeig directe de la textura a tot el conjunt de meshes per tal de solucionar aquests problemes. Tot i així, no es va arribar a
implementar aquesta idea per la complexitat d’aquest algoritme al tenir varies meshes ja que haurien d’estar unides entre elles d’alguna forma. A més, no es podia fer un mapeig de la textura sola, ja que mapejar una sola textura a tot el conjunt format per la cova hauria provocat que es veies en molt baixa resolució, per tant es va descartar.

Figura 42: Degut als problemes esmentats, al texturar un forat es pot arribar a veure com si la textura estigues comprimida, degut als valors d’interpolació. Si es sumen els problemes del suavitzat, fa que es vegi encara més poligonal i amb pitjor definició

14.12.3 Tancament de túnel

Per tal de texturar el tancament del túnel es van tenir en compte dues alternatives, ambdues amb els seus pros i les seves contres:

- Utilitzar com a coordenada de textura del baricentre una igual a la coordenada de textura y de la polilínies més el radi mínim de la polilínies, i amb una x final centrada, és a dir 0.5. S’obté un resultat molt bo excepte entre l’últim i primer vèrtex, degut a què es com si s’estigues agafant un triangle de la textura, i per tant queda un espai molt gran entre aquests dos vèrtexs.

- Utilitzar tants vèrtexs com tingui la polilínies, tots amb una posició igual a la del baricentre, però amb diferent coordenada de textura. La coordenada de textura es generaria de la mateixa forma que si s’estigués fent una extrusió normal, amb una distància igual al radi mínim. El problema d’això es que al tenir tots aquests vèrtexs la mateixa posició a l’espai, es
Figura 43: Al estar tots els vèrtexs de la última polilínia d’un túnel (blau) triangulant amb un vèrtex amb coordinada de textura $x = 0.5$, quedarà un tros sense texturar correctament entre l’últim i primer vèrtexs, degut a la gran diferència d’espai (representat com a gris a la figura).

Figura 44: Es pot apreciar com al estar triangulant amb el mateix vèrtex amb coordinada de textura $x = 0.5$, entre l’últim i primer vèrtex es texturarà incorrectament, formant una línia visible com si alguns dels triangles no s’estiguessin pintant i per tant es nota la falta d’aquest triangles, com es pot observar a la figura 45. Tot i així, per veure aquest efecte s’ha de mirar la triangulació des de molt a prop. A més, segons quina textura s’utilitzi, aquest efecte es pot apreciar més o menys.

Finalment es va decidir quedar-se amb la segona com a definitiva, ja que l’usuari s’ha de fixar molt per veure el seu defecte, no com la primera que és molt fàcil adonar-se.
Figura 45: Com que realment els vèrtexs de la polilònia del tancament de túnel (f, verd) comparteixen una única posició, quan es trianguli entre dos vèrtexs d’aquesta i un de última polilònia del túnel (a, blau), serà com si aquest triangle no existís, creant un buit (representat com a gris a la figura) que es notarà al apropar-se

Figura 46: Al texturar amb n vèrtexs, tot i que de lluny no es nota, al apropar-se al tancament del túnel es poden apreciar els trossos de textura que faltarien

14.12.4 Espeleotemes

Tot i que s’hauria obtingut millors resultats utilitzant una textura diferent per als espeleotem, es va decidir utilitzar la mateixa que per la cova. Es van plantejar dos solucions per a aquesta texturització:

- El procediment inicial que es seguia era molt semblant al de quan es fa una extrusió: A partir de les coordenades de textura dels vèrtexs triats per a crear l’espeleotema, a mesura que s’anava extrudint aquest, s’anava
aplicant un increment en y dels vèrtexs. El problema d’ajó és que, al formar els vèrtexs un rectangle a la textura, hi havia resultats que es veien malament al haver d’interpolejar entre dos posicions que tenien la mateixa coordenada Y.

- Per tal de solucionar el problema esmentat, es segueix el mateix procediment anterior, però es va decidir que l’increment de textura a una extrusió tingú també un valor a x, de forma que entre els parell de vèrtexs de la polilínia original amb la mateixa coordenada y, tinguessin diferent coordenada y que la seva extrusió O.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure47.png}
\caption{Comparació d’incrementar a una extrusió d’un espeleotema les coordenades de textura, entre la polilínia a (verda) i la b (blava).}
\end{figure}

Figura 47: Comparació d’incrementar a una extrusió d’un espeleotema les coordenades de textura, entre la polilínia a (verda) i la b (blava). Es pot observar com al formar un quadrat en l’espai de textura, incrementar només en la direcció de les y provocarà que al interpolar entre \( a_1, a_2 \) i \( b_1, b_2 \) no s’obtindrà cap diferència en les y. En canvi, si s’afegeix també un increment en x si que hi haurà diferència i per tant es texturarà correctament.

Per tant, el procediment a aplicar en aquest cas seria el mateix que a una extrusió, però posant el valor de \( \text{Increment}UV \) a \( (\text{Distance} / \text{UVfactor}, \text{Distance} / \text{UVfactor}) \) si es tracta d’una estalagmita, o a \( (\text{Distance} / \text{UVfactor}, -\text{Distance} / \text{UVfactor}) \) si es tracta d’una estalactita.

### 14.12.5 Obtenció de textures

Les textures s’han obtingut de dues planes webs diferents: Filter forge i CG-Textures. S’ha triat diferents tipus de textures ja que això permet l’immersió d’estar en diferents medis.
Figura 48: Exemple d’espleotema texturat

A part de les textures de color, també s’ha fet ús dels mapes de normals\textsuperscript{19} per tal d’afegir rugositat i obtenir així un millor aspecte.

Aquest material prèviament creat seria l’únic que s’utilitza i no ha estat generat de forma procedural.

\textsuperscript{19}Els mapes de normals són un tipus de textures utilitzats per a especificar la normal utilitzada per calcular la il·luminació en una certa coordenada de textura, rebent aquesta tècnica el nom de \textit{Bump mapping}. 

105
Figura 49: Exemples de les diferents textures aplicades. Es pot observar que en algunes es nota més l’efecte de texturització als forats que a altres, així com les normals diferents al canvi de túnel.

14.13 Il·luminació

La il·luminació és també un apartat secundari per al projecte, però fonamental per a la visualització ja que és el que permet que el color es vegi d’una forma o una altre, de forma que sense il·luminació es veuria tot negre. Les diferents implementacions per a la il·luminació s’exposen a continuació:

- Inicialment, només es tenia la Directional Light per defecte de Unity, que vindria a ser una llum que simula el punt de llum del sol. Degut a què ha de tenir una direcció a la qual està il·luminant, hi havia parts de les coves que es veien molt fosques.

A continuació es va voler generar diferents punts de llum com si fos una
operació d’extrusió més. El principal problema que tenia això es que es creaven masses i afectava molt negativament al rendiment de la renderització.

- Es va considerar com a alternativa crear només punts de llum que estiguessin al voltant del personatge, i que s’anessin adaptant a la mida de la cova per on estava el personatge. No es va arribar a implementar aquesta alternativa degut a què era massa complicada pel que oferia, ademés d’esperar-se no obtenir uns resultats massa bons.

- D’aquesta alternativa, però, va derivar la que finalment s’utilitza al programa i dá la les resultats bastant satisfactoris, ademés de ser relativament simple: Consta en colocar un punt de llum darrere del personatge i una llum en àrea davant com si fos una llanterna. Aquest aspecte afegeix immersió de cara al jugador a estar realment explorant una cova o mina. Addicionalment, es treu la llum direccional ja que d’aquesta forma dona la impressió d’estar a un lloc fosc on no arriba la llum solar.

14.14 Moviment i càmera

El personatge a controlar i la càmera que el segueix és l’apartat en el que menys s’ha treballat al considerar-se un objectiu secundari, degut a què l’enfoc del projecte era la generació en si de la cova o mina. Per tant, amb un moviment del jugador i una càmera que el seguís de forma simple ja n’era suficient, ja que ja permetia un mínim d’exploració.

- Primer es va crear un script simple de moure al jugador, el qual era una esfera sense restriccions de gravetat per tal de poder explorar el resultat de la generació més lliurement. Hi havia botons que permetien avançar cap endavant i endarrere o cap dalt i cap abaix, així com la possibilitat de girar la càmera respecte l’eix Y de forma com si estés mirant als costats. Degut a què per detectar les col·lisions s’utilitzava el propi sistema de físiques de Unity sense cap modificació, hi havia la repercussió dels xocs als col·lisionadors, de forma que s’anava acumulant la força d’aquests xocs. A més a més, en moltes ocasions la càmera s’allunyava de forma que es situava a l’exterior a la cova.

- Degut a què tot això complicava bastant l’exploració i s’allunyava de l’objectiu de ser simple, es va decidir utilitzar la pròpia Multipurpose camera
de *Unity*, la qual solucionava molts casos on la càmera s’allunyava de la cova degut a què utilitza *scripts* on s’intenta evitar que no hi hagin obstacles entre el jugador i la càmera. Tot i així segueixen havent-hi alguns casos on s’allunya i enfoca part de l’exterior de la cova. Respecte al personatge a controlar finalment, es va procedir a crear un personatge propi amb una eina web externa\(^\text{20}\) i importa-lo a *Unity*, així com importar models gratuïts de la *Unity Asset Store*, utilitzant l’esquelet i les animacions que venen per defecte amb el personatge característic de *Unity*.

Figura 50: Exemples dels diferents personatges provats. Els dos primers han estat creats amb una eina externa, mentre que els altres dos s’han obtingut de la *Unity Asset Store*. S’ha agafat com a definitiu el mag.

Aquesta combinació s’ajusta a voler tenir un moviment simple que permet l’exploració, que ja és l’objectiu. L’únic problema que existeix, es que per a calcular el col·lisionador de la cova, s’utilitza directament les pròpies

\(^{20}\text{Autodesk Character Generator, una eina gratuïta la qual permet crear models tridimensionalis bipèdes de forma molt sencilla}\)
meshes que representen cada túnel. Això, a part de tenir el problema de què per a una cova molt gran fer el càlcul de les col·lisions seria bastant costós, provoca que en algunes ocasions el personatge es comporti de forma estranya.
15 Treball futur

A continuació s’esmenten algunes millores que s’han pensat per a millorar el resultat de la generació, i no s’han pogut implementar per la seva complexitat i falta de temps:

- Poder generar més d’un forat per extrusió. Es podria fer de forma que es generés un forat simètric al que ja s’ha creat, comprovant que no s’estigués agafant cap vèrtex de l’altre forat.

- Millorar texturització als forats. Com s’ha comentat, és el pitjor aspecte de la visualització, però per tal de solucionar-ho caldria aplicar un altre tipus de texturització per tota la cova, pel que caldria replantejar-se tots els aspectes d’aquesta.

- Irregularitat: Extrudir només alguns dels vèrtexs de la polilínia cap a dins o cap enfora. Això afegiria irregularitat al model creat. Tot i així, es deixaria de complir algunes de les propietats de la polilínia esmentats, pel que caldria replantejar tot l’algoritme.

- Possibilitat de què l’usuari seleccioni punts d’interès a la cova: com per exemple punts a forçar la cova a passar per allà, punts on es faci spawn d’enemics o objectes varis pel joc, o punts on es forci la cova a aplicar-se alguna de les operacions, com per exemple escalar per tal de què la cova es faci més o menys ample.

Tot i no haver-se implementat això, es va pensar com es podria fer. Una de les formes pels punts on es forçaria a passar la cova seria generar un graf connex i sense intersecció amb tots els punts, tenint en compte la polilínia inicial com un punt més. S’utilitzaria aquest graf per, a partir de la polilínia inicial anar extrudint i fent forats segons es vagin trobant amb els diferents punts del graf, com si es volgués seguir un camí. Per tal de fer això caldria crear un algoritme per tal de generar el graf esmentat. A partir de fer la generació d’aquest forma, es podria prendre com una base i a partir d’aquí extrudir i aplicar altres operacions aleatoriament. Alguns dels problemes que podrien haver-li seria el no existir cap graf amb les propietats esmentades, o bé a l’hora de generar la cova-

- En comptes d’evitar interseccions, fer un forat a la zona on s’anava a interseccionar i juntar-lo amb l’extrusió actual. Això, tot i obtenir un model
molt més semblant als models creats manualment al permetre que diferents bifurcacions s’ajuntin, es considera una tasca molt complexa, de fet la més complexa de tot el que s’ha arribat a plantejar. La seva complexitat prové en que caldria destriangular la zona on s’anés a interseccionar. Això implicaria primer tant trobar la zona corresponent com eliminar els triangles creats a aquella zona, i per fer això en un temps computacional acceptable s’hauria de fer ús segurament d’octrees, com de una estructura de dades que permetés trobar ràpidament els triangles corresponents a destriangular. L’únic que es podria fer de forma immediata tal i com està el projecte, seria retriangular entre els dos túnels: Amb el que s’està a punt d’interseccionar i l’actual.

A més, també caldria tenir en compte que d’aquesta forma existeix el cas amb el que s’està a punt d’interseccionar amb més d’un túnel, i això complicaria encara més les coses.
16 Conclusions

L’àrea dels videojocs en la informàtica és un tema pel qual sempre he estat interessat a nivell personal. Per aquesta raó, vaig voler escollir un treball de fi de grau relacionat amb això. Tot i al principi no estar del tot convençut amb un tema relacionat amb la generació procedural al no saber-hi del tema ni haver-hi treballat abans, puc concloure que he obtingut diversos coneixements gràcies a tot l’aprés durant el grau, especialment en l’àrea de gràfics i geometria.

L’haver fet un treball d’aquestes dimensions, amb pocs llocs on poder fer recerca al ser un mètode no existent prèviament, m’ha permès madurar en quant a l’anàlisi d’algoritmes i estructura de codi, un aspecte essencial quan es vol desenvolupar software.

La realització del treball m’ha ajudat també a adonar-me de la importància de tenir un feedback constant i en ser flexible a l’hora de aplicar canvis i millores al projecte, en comptes de cenyir-se a una planificació poc real.

Un dels aspectes del projecte que més m’hauria agradat millorat hauria estat la visualització final dels forats. Tot i així, em sento bastant orgullós amb tot el treball fet i els resultats obtinguts, així com el compliment dels objectius no només principals, sinó també secundaris.

Tot i no haver estat possible per falta de temps, hauria estat molt interessant pujar el projecte a la Unity Asset Store, ja que d’aquesta forma hauria permès no només el seu ús per altres usuaris, sinó també la possibilitat de poder-li aplicar les millores proposades. De totes formes, no es descarta el fer-ho en el futur i, gràcies a la forma en la que s’ha implementat, facilita aquest aspecte de possible millora.
Quant a la prospectiva d’aquest treball, un cop aplicades aquestes millores proposades, entre d’altres, crec que es podria obtenir una bona eina per poder utilitzar en la generació de coves en un videojoc comercialitzable.

Finalment, m’agradaria agrair al director del projecte per tot el seu gran re-
colzament i disponibilitat durant el desenvolupament del projecte, especialment per el seu interès a l’hora de voler tutoritzar-me el projecte.
També vull agrair a totes les persones de l’àmbit personal per la seva paciència i ànims durant tots aquests anys per ajudar-me a seguir endavant i no tirar la tovallola.
17 Documentació per usuari

Generar la cova és molt simple: simplement cal donar-li al *Play de Unity*, i acte seguit definir com serà la forma aproximada de l’entrada de la cova seleccionant un conjunt de punts. Aquests punts s’aniran unint entre ells fins finalment seleccionar l’últim punt, que s’unirà amb el primer. És important que mentre es seleccionin els punt no es formin polilínies amb formes estranyes, com per exemple entrecreuaments, ja que això podria produir resultats no desitjables. El número de punts a seleccionar, entre tots els altres paràmetres disponibles per al programa s’expliquen al següent subapartat.

És important destacar que aquests paràmetres també poden ser controlats via codi al ser variables públiques dels respectius *scripts* parametritzables. D’aquesta forma, es podria utilitzar aquest projecte com a una eina addicional i controlant els paràmetres externament.

17.1 Paràmetres

Només hi ha dos *scripts* amb valors parametritzables per tal de simplificar-ho tot. El primer és el *Cave Generator* que com el seu propi nom indica conté els paràmetres bàsics per a la generació:

- **Cam**: La càmera a partir de la qual es podrà seleccionar els punts d’entrada de la cova.
- **Gate Size**: Mida de l’entrada de la cova sense suavitzar, així com de l’inici de qualsevol túnel.
- **Max Holes**: Màxim de forats que es podran fer, i per tant representarà també el número màxim de túneles.
- **Max Extrude Times**: Màxim de extrusions que es poden fer per cada túnel, de forma que tots els túneles comparteixen aquest valor.
- **Smooth It After Generation**: Número d’iteracions que s’aplicaran per al suavitzat després de la generació als vèrtexs dels túneels seleccionats per a crear un forat. A partir d’un valor de 50 aproximadament, deixa de tenir sentit incrementar aquest valor. Cada iteració té un cost igual al número de vèrtexs marcats per el número de vèrtexs adjacents a cadascun.
- **Smooth It Ini Tunel**: Número d’iteracions que es
• **Method**: Mètode de generació que es farà servir, amb unes opcions autodescriptives: *Recursive*, *IterativeStack* i *IterativeQueue*

• **Player**: Prefab del personatge que el jugador farà servir. Aquest ha de contenir la seva pròpia càmera.

• **Cave Material**: Material que s’aplicarà per tota la cova, per tal de poder texturar-la.

• **Show Generation**: Booleà que indica si es mostrarà la generació de la cova pas a pas un cop s’executi.

L’altre script amb valors parametrizables és **DecisionGenerator**, amb un nom també bastant informatiu: té paràmetres que serveixen per decidir la freqüència i valors de les diferents operacions a aplicar a les extrusions:

• **Direction Extr Base**: Valor base que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació de canvi de direcció.

• **Direction Extr Desv**: Valor de desviació que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació de canvi de direcció.

• **Scale Extr Base**: Valor Base que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació d’escalat.

• **Scale Extr Desv**: Valor de desviació que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació d’escalat.

• **Rotation Extr Base**: Valor Base que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació de rotació.

• **Rotation Extr Desv**: Valor de desviació que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació de rotació.

• **Stalgm Extr Base**: Valor Base que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació de generar espeleotemes.

• **Stalgm Extr Desv**: Valor de desviació que s’agafà per a decidir durant quantes extrusions s’aplicarà l’operació de generar espeleotemes.

• **Direction K Base**: Valor base que s’agafà per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de canvi de direcció.
• **Direction K Desv**: Valor de desviació que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de canvi de direcció.

• **Scale K Base**: Valor base que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació d’escalat.

• **Scale K Desv**: Valor de desviació que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de canvi d’escalat.

• **Rotation K Base**: Valor base que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de rotació.

• **Rotation K Desv**: Valor de desviació que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de rotació.

• **Stalgm K Base**: Valor base que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de generar espeleotemes.

• **Stalgm K Desv**: Valor de desviació que s’agafa per a decidir durant quantes extrusions s’ha d’esperar per a tornar a generar l’operació de generar espeleotemes.

• **Distance Small Min**: Valor mínim del rang al generar una distància petita, al generar una extrusió.

• **Distance Small Max**: Valor màxim del rang al generar una distància petita, al generar una extrusió.

• **Distance Big Min**: Valor mínim del rang al generar una distància gran, al generar forats.

• **Distance Big Max**: Valor màxim del rang al generar una distància gran, al generar forats.

• **Direction Y Walk Lim**: Valor de la component y que pot tenir com a màxim la direcció d’extrusió, per tal de què no hi hagin pujades o baixades massa inclinades.
• **Direction Max Angle**: Límit de la diferència que pot haver-hi entre la direcció actual i la generada aleatòriament.

• **Scale Limit**: Límit que pot tenir el valor generat aleatòriament per a aplicar un escalat.

• **Rotation Limit**: Límit que pot tenir el valor generat aleatòriament per a fer una rotació.

• **Hole K**: Paràmetre utilitzat per a la decisió de quan fer un forat, explicat a continuació.

• **Hole Prob**: Paràmetre utilitzat per a la decisió de quan fer un forat, explicat a continuació.

• **Hole Lambda**: Paràmetre utilitzat per a la decisió de quan fer un forat, explicat a continuació.

• **Hole Condition**: Indica la condició per fer un forat. Pot prendre els següents valors:
  
  – **EachKDesv**: Cada HoleK extrusions s’intentarà generar un forat.
  – **EachKDesvProb**: Cada HoleK extrusions s’intentarà generar un forat amb probabilitat HoleProb
  – **MoreExtrMoreProb**: Com més extrusions s’hagin produït a un túnel, més probabilitats d’intentar generar un forat hi hauran, començant amb HoleProb i donant a cada extrusió una contribució de HoleLambda a la probabilitat.
  – **MoreExtrLessProb**: Com més extrusions s’hagin produït a un túnel, menys probabilitats d’intentar generar un forat hi hauran, començant amb HoleProb i donant a cada extrusió una contribució de HoleLambda a la probabilitat.

• **Hole Min Vertices**: Número de vèrtexs mínims que un forat ha de tenir.

• **Hole Max Vertices**: Número de vèrtexs màxims que un forat pot tenir.

• **Holes Max Angles Direction**: Utilitzat a l’hora de decidir quins vèrtexs formaran part d’un forat, indica l’angle màxim que pot haver-hi entre un vèrtex que serà seleccionat per pertanyer al forat i la direcció aproximada que tindrà el forat.
18 Glossari

1. **Roguelike**: Gènere de videojocs nascut als anys 80 on l’acció transcorre a una masmorra o lloc semblant el qual ha estat generat de forma procedural.

2. **Octree**: Estructura de dades molt utilitzada per representar subdivisions a l’espai.

3. **Unity Asset Store**: Tenda oficial del software Unity on hi ha diferents assets o projectes creats per usuaris d’aquest software. Poden ser gratuïtes o de pagament.

4. **Artifact**: Polígons els quals interseccionen amb ells mateixos, entre altres errors que provoquen un resultat poc agradable a la vista.

5. **Scrum**: Metodologia àgil on els cicles de desenvolupament son incrementals en comptes de planificats, així com el solapament de diferents tasques en comptes de fer-les de forma seqüencial.

6. **Git**: Software de control de versions molt utilitzat a projectes softwares, per tal de facilitar la visualització de canvis, entre altres coses.

7. **GitHub**: Plataforma per allotjar projectes utilitzant el control de versions Git.

8. **Bug**: Error al comportament d’un software creat degut a un problema al codi, que provoca resultats diferents als esperats.

9. **Open source**: Software lliure de pagament pel seu ús i la seva distribució, així com la possibilitat de modificar-lo.

10. **Z-fighting**: Efecte que es produeix en els gràfics per computador quan dos triangles diferents tenen en coordenades de la càmera una z molt semblant, per el que es va intercalant quin dels dos triangles es pinta davant a mesura que es va movent la càmera, produint un efecte poc agradable.

11. **Mesh**: Malla de triangles que representa un objecte tridimensional. És l’estructura de dades que es fa servir per guardar tota la informació d’un model 3D.

12. **Distribució Gaussiana**: També anomenada distribució normal, és una distribució probabilística on té forma de campana i és simètrica respecte
el punt d’inflexió de la campana, on la probabilitat és més gran. Molts fenòmens reals la segueixen.

13. **Distribució Exponencial**: Distribució probabilística que segueix una funció exponencial.

14. **Render**: Acció de pintar tots els conjunts de triangles d’una escena en l’ordre adequat, així com descartar els innecessaris i que per tant no caldrà pintar.

15. **Tile**: Textures rectangulars o quadrades que mostren continuïtat pels costats. Molts jocs 2D estan basats en aquest tipus de textures.

16. **Spwan**: Acció que poden fer per exemple els enemics o objectes d’un videojocs, i fa referència a quan apareixen a l’escena. Així doncs, un punt de **spawn** seria un lloc on es generen elements del joc.
Referències


Apèndix A  Desglòs de la dedicació en hores

<table>
<thead>
<tr>
<th>Fase</th>
<th>Activitat</th>
<th>Disseguida (h)</th>
<th>Disseguida dur</th>
<th>Desviació dur</th>
<th>Desviació HW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figuents</td>
<td>Delegat</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Plannig</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Gerència</td>
<td>6</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Intemperies</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Presentació</td>
<td>12</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Defensa</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Preparació</td>
<td>10</td>
<td>0</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Inici de l'actuació</td>
<td>20</td>
<td>70</td>
<td>10</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>Preparació d'intemperies</td>
<td>20</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Analisi i implementació Generator</td>
<td>360</td>
<td>58</td>
<td>83</td>
<td>126</td>
<td>380</td>
</tr>
<tr>
<td>Analisi i implementació extras</td>
<td>50</td>
<td>12.5</td>
<td>19.5</td>
<td>19.5</td>
<td>50</td>
</tr>
<tr>
<td>Final</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>

Taula 9: Desglòs de la planificació temporal per tasques i rols, així com la dedicació real i la desviació
Apèndix B Desglos del pressupost

Taula 10: Desglos del pressupost per tasques i conceptes, així com el cost real i la desviació. Les hores dels diferents rols s’han obtingut a partir de l’apèndix A, mentre que pel hardware s’ha utilitzat les hores totals per cada tasca.