



**Treball de Fi de Grau**  
**GRAU D'ENGINYERIA INFORMÀTICA**  
**Facultat de Matemàtiques**  
**Universitat de Barcelona**

---

**ALGORITMES PROCEDIMENTALS PER  
AL DESENVOLUPAMENT D'ENTORNS  
ALEATORIS: CREANT UNA ILLA**

---

**Marc Anglés Castillo**

Directora: Anna Puig Puig

Realitzat a: Departament de  
Matemàtica  
Aplicada i Anàlisi.  
UB

Barcelona, 27 de juny de 2015

# ÍNDEX

---

1	Introducció	4
1.1	Àmbit del projecte	4
1.2	Motivació	5
1.3	Objectius Generals	5
1.4	Objectius Específics	6
1.5	Organització de la memòria	6
2	Antecedents	6
2.1	Resum dels treballs més recents	6
2.2	Treballs relacionats amb illes	8
2.3	Conclusió	11
3	Anàlisi	11
4	Disseny	11
4.1.1	Generació de punts aleatoris	12
4.1.2	Generació de soroll	13
4.1.3	Detecció dels biomes	14
4.1.4	Aplicació de les textures i il·luminació final	15
4.2	Optimització	17
4.3	Interfície de l'usuari	17
4.3.1	Posicionament	18
4.4	Diagrames de classes	20
4.5	Classes	21
4.5.1	Island.cs	21
4.5.2	HeightmapGenerator.cs	21
4.5.3	Blob.cs	22
4.5.4	TextureScale.cs	22
4.5.5	Soroll.cs	22
4.5.6	ExportTerrain.cs	23
4.5.7	Camara.cs	24
4.5.8	UIController.cs	24
5	Resultats i Simulacions	25
5.1	Funcionament de l'aplicació	25
5.2	Proves de rendiment	28
5.2.1	Incrementant la mida del mapa	28
5.2.2	Incrementant el número de blobs	29
6	Valoració econòmica	30
6.1	Anàlisi del temps de realització del projecte	30
6.2	Valoració del cost econòmic del projecte	30
7	Conclusió	31
8	Bibliografia	33
	Apèndix	34

## Abstract

*This project is aimed to answer an inner enigma some players can have. How in the world do games like Minecraft or Diablo create a different and amazing world every time I tune in?*

*The answer is, as most developers may know, procedural generation and programming. This program is more focused on the terrain generation, while procedurality is aimed at everything in this world and can create creatures, plants, buildings, even gameplay variations.*

*But this kind of programming has got a clear and hard rock limitation, it is extremely complex to setup if the structure created is complex and the easy to configure algorithms generate poor results.*

*While developing all this code an project, and through all this paper, it will be seen that it es fairly easy to generate easy structures without hard mathematic work but realistic ones require a bit more than coding hard.*

Aquest projecte te com a objectiu respondre una pregunta que es fan molts jugadors. ¿Cóm s'ho fan els jocs com Minecraft o Diable per a generar nous mons cada vegada que carrego partida?

La resposta, com molts desenvolupadors ja deuen saber, es la programació per procediments. Aquest programa esta enfocat a la generació de terrenys, mentre que la proceduralitat, en general, es pot fer servir per a generar de tot, com plantes, edificis o fins i tot variacions de gameplay.

Però aquest sistema de programació té una limitació clara. Els algoritmes que produeixen contingut interessant i realista son realment complexos de generar, i els que no son tant difícils generen contingut pobre.

Desenvolupant aquest codi i a través de la memòria, es veura que es relativament senzill generar estructures simples, pero que sense una enorme base matemàtica no es poden assolir grans estructures.

# 1 INTRODUCCIÓ

---

Ja fa uns anys que la creació de mons virtuals ha avançat cap a la generació aleatòria dels escenaris on es realitzen les diferents interaccions; des de **Minecraft** fins a **The Binding of Isaac**, són molts els títols que aposten per aquest sistema per produir experiències úniques per a cada usuari, sense haver d'invertir grans quantitats en disseny. Tots aquests sistemes es troben dins del paradigma de la programació procedimental.

L'objectiu d'aquest treball és desenvolupar un mètode procedimental per a la generació d'una illa de forma aleatòria però coherent, podent alterar un conjunt de paràmetres que permetin l'alteració de característiques, tals com l'altitud sobre el nivell del mar, superfície, rugositat del perfil, etc.

El mencionat projecte serà portat a terme en el llenguatge de programació C# fent servir Unity per a analitzar els resultats del mètode proposat i per a modelar la interfície d'usuari.

## 1.1 ÀMBIT DEL PROJECTE

Encara que la temàtica del treball es centri en el camp de l'algorísmica, el fet que el resultat final sigui aplicat amb una representació tridimensional ens apropa també a la visualització de dades per computador.

En el transcurs d'aquest grau ens hem trobat amb múltiples assignatures que ajuden a la comprensió d'aquesta tasca. Les primeres i més òbvies són, potser, Algorísmica i Algorísmica Avançada, ja que gran part del projecte es basa únicament en la capacitat del programador per a resoldre certs problemes amb mètodes eficients.

També la matèria de Gràfics per Computador ajuda en el moment que es comencen a tractar els models 3D; el coneixement de l'estructura d'un d'aquests models a nivell teòric ha permès manipular-los lliurement via codi, tot i que l'aproximació de Unity és molt més senzilla de tractar que un sistema complet de motor gràfic, com pot ser OpenGL o DirectX.

Finalment, alguns dels problemes presentats haurien pogut ser solucionats per la via de la paral·lelització i concurrència, però per a no desviar massa el projecte i no entrar en camps que no són propis de la idea original, s'ha preferit deixar-ho com a possibles ampliacions.

## 1.2 MOTIVACIÓ

La idea d'aquest projecte té origen en una qüestió oberta: quin és el sistema de **Minecraft** a l'hora de generar es seus paisatges?

**Minecraft** és un videojoc en el qual els usuaris es troben amb mons voxelics generats de forma completament procedimental, de manera que cadascun dels jugadors viu una experiència única i totalment nova amb pràcticament infinites variacions. Però aquests mons no són aleatoris, ja que tot en ells té una relativa coherència.

I no només **Minecraft**, són molts els títols que opten per la generació procedimental com a marca insígnia per oferir molt més contingut del que el seu pressupost els permetia. Un dels exemples més importants es, probablement, **No man's sky**, producte encara en desenvolupament on l'usuari viatja de planeta en planeta dins d'una galàxia a mida real en la qual cada astre i localització s'ha desenvolupat de la manera descrita, en el moment del joc.

No obstant això, en molts casos aquests algorismes són pobres i limitats, i poden no oferir una experiència equilibrada de joc.

## 1.3 OBJECTIUS GENERALS

El projecte planteja el desenvolupament d'un algorisme de creació d'entorns aleatoris. Donat que aquest camp es molt ampli, es centra en la generació de terrenys, concretament illes. Aquesta decisió permet concretar més el problema, ja que no es té en compte el tipus de frontera ni la possibilitat de múltiples estructures de terreny específiques.

## 1.4 OBJECTIUS ESPECÍFICS

- Desenvolupar un sistema de blobs per a generar escenaris
- Trobar una forma de determinar biomes
- Investigar el sistema de modelat més òptim

## 1.5 ORGANITZACIÓ DE LA MEMÒRIA

Antecedents: Petita definició del que es pot trobar al món del desenvolupament procedimental. Algoritmes coneguts, entorns d'aplicació i treballs d'exemple.

Anàlisi: requeriments que ha de tenir aquest projecte i limitacions a l'hora de desenvolupar.

Disseny: Procés complet de desenvolupament del programa. Situacions complicades amb les que s'ha hagut de treballar i solucions a aquestes situacions.

Diagrames: totes les representacions gràfiques necessàries per a entendre el funcionament del programa.

Resultats i simulacions: Explicació de l'aplicació final i proves de rendiment.

Conclusió: Raonament final del projecte.

# 2 ANTECEDENTS

---

## 2.1 RESUM DEL TREBALLS MÉS RECENTS

A l'article **A survey on Procedural Modeling for Virtual Worlds** [mem1] pot apreciar-se que la necessitat de generar estructures de tot tipus de forma procedimental va néixer fa més de trenta anys; això es deu a la àmplia varietat de models que poden ser creats amb aquest mètode: textures, plantes, terreny, edificis, àrees urbanes, xarxes viàries, conques fluvials, etc.

La principal raó d'aquest creixement es troba en la Amplificació de Dades; a partir d'un petit set de dades o conjunt de paràmetres, es pot obtenir una font enorme de contingut.

Una altra de les característiques importants és la Compressió de Dades, on mitjançant un conjunt petit de paràmetres es poden generar nous models, dades molt més senzilles de desar que el format del model mateix. Però aquesta generació sovint és estocàstica, raó per la qual aquest conjunt de dades només definirà unes característiques que crearan un set de models amb les mateixes qualitats, però tots diferents entre sí.

En el context de la generació de mons virtuals es troben molts sistemes i algorismes que, de formes diferents, permeten crear el que es necessita.

## **TERRENY**

La forma més habitual de representació d'un terreny és la utilització de mapes d'alçades (*Height Map*), una matriu de dues dimensions on cada cel·la representa l'elevació al terreny original. Aquest mapes són fàcils de comprimir i manipular per la GPU, però es veuen limitats al no poder representar estructures sortints ni coves. Per a solucionar aquest problema es van crear les Estructures per Capes de Dades (*Layered Data Structures*), els Vòxels i Mesh 3D.

Finalment Peytavie [mem2] ha desenvolupat un model que permet la representació de sortints, diferent materials i roques soltes.



Els primers algorismes de generació de *heightmaps* introduïen una porció controlada d'aleatorietat per a generar una elevació detallada. El primer algorisme de subdivisió en aparèixer va ser conegut com *Midpoint Displacement Method* [mem3] en el què l'altura d'un punt és calculada amb la mitjana dels seus veïns en forma de triangle o diamant.

Hi ha altres mètodes estocàstics, com per exemple els de generació de soroll, com el *Soroll de Perlin*, i variacions d'aquest, anomenats *Moviment Brownià* que van bé per generar crestes o petits turons.

Tot i així, els anomenats mètodes es veuen limitats pel fet que els paràmetres utilitzats no són intuïtius, i pot fer que generar el contingut desitjat sigui una mica més difícil del que hauria de ser. No obstant, en contraposició, el rendiment del soroll del *Moviment Brownià* és extremadament alt i fàcil d'optimitzar i paral·lelitzar, ja que el valor de cada punt no depèn dels seus veïns.

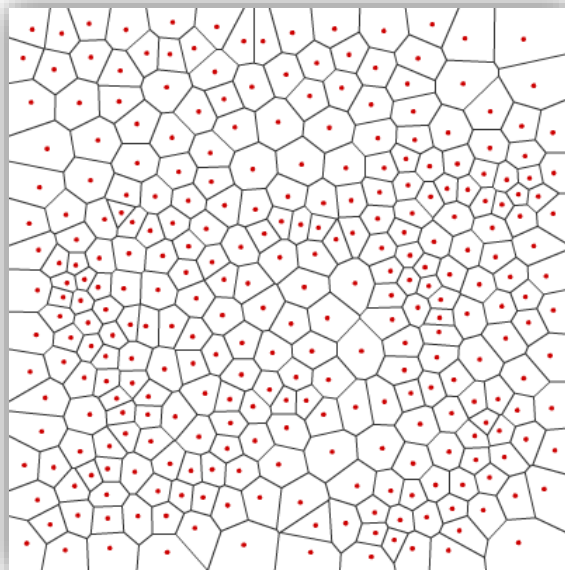
Zhou *et al.* [mem4] van desenvolupar un algorisme que genera terrenys a partir de línies dibuixades per l'usuari, que poden definir crestes o d'altres elevacions contínues. L'input és intuïtiu però el mètode no està recomanat si el que es vol és el control sobre petites unitats de terreny. A més a més, el sistema, com tots els mètodes basats en exemples, està limitat pels conjunts d'exemples desats.

També trobem altres sistemes més específics que fan servir dades introduïdes per l'usuari, com regions delimitades amb un dibuix, o que serveixen per a generar un sol tipus de terreny senzills com pot ser una muntanya, però amb molt de detall i facilitat de parametrització.

## 2.2 TREBALLS RELACIONATS AMB ILLES

Un grup d'alumnes de la universitat de Stanford ha creat un projecte de generació aleatòria d'illes.[mem5]

En ell, es fa una aproximació matemàtica molt complexa, i en dues dimensions, que crea un mapa d'alçades molt complex però realista. L'algorisme comença generant un nombre concret de centroides i els fa servir per dibuixar polígons de Voronoi. Per a evitar que sigui tot massa aleatori, aquests centres són generats mitjançant l'algorisme de Lloyd.

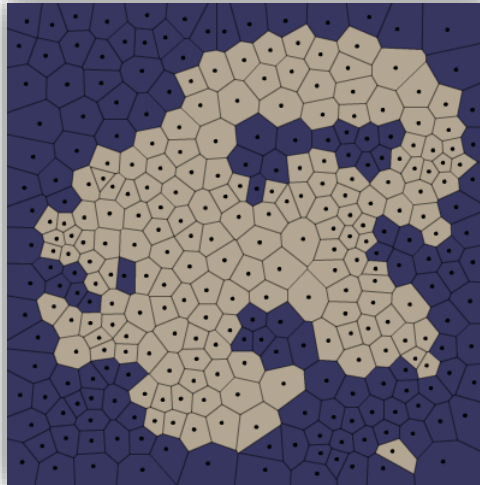




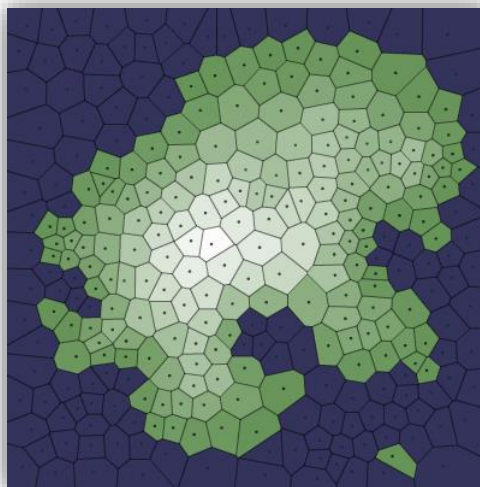
A partir dels centres i arestes es generen dos grafs, un d'arestes i l'altre de centres, sent aquest últim el que es fa servir com a perfil i cel·les de l'illa.

Un cop seleccionada la forma de l'illa (quines cel·les seran aigua i quines terreny), es generen tres mapes diferents:

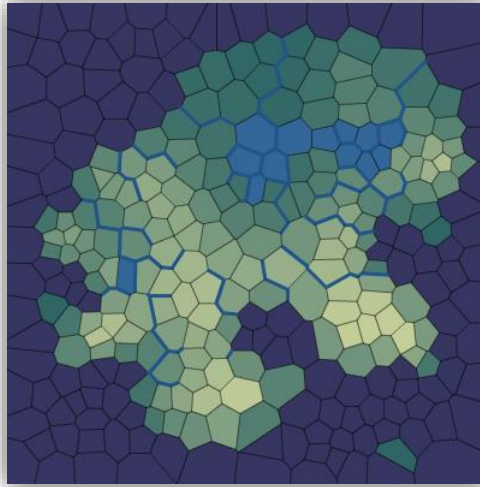
1. **MAPA DE TERRENY:** Terra, aigua interna i mar. El desenvolupador ho fa amb sinus, blobs, *Soroll de Perlin* o siluetes entrades manualment.



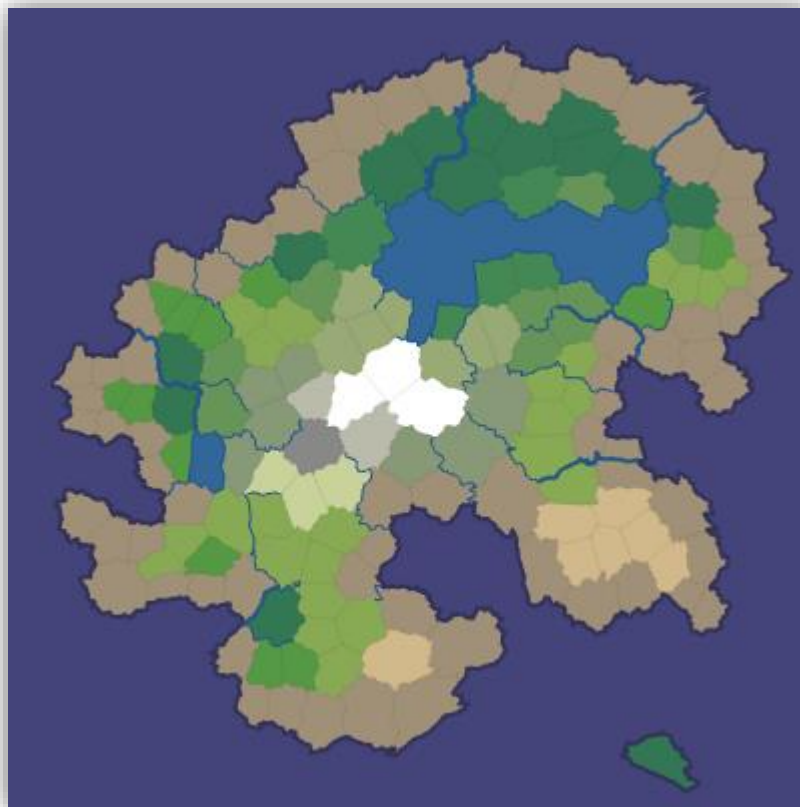
2. **MAPA D'ALÇADES:** Generat pel terreny llunyà a la costa, tant el més alt i proper com el més baix. A partir d'aquí, i seguint lògiques d'alçada (arestes que equivalent a baixants cap al mar) es poden prendre algunes arestes per a generar un riu.



3. **MAPA D'HUMITATS:** Finalment, aquest sistema es genera amb la proximitat a aigua. Com més distant, menys humitat, i es fa servir per a poder assignar biomes.



Quan s'apliquen els tres apartats s'aconsegueix una illa completament generada, a la qual sempre s'hi podem afegir millores, com una representació més realista del mapa mitjançant soroll, generació de carreteres i camins i creació de centres de població entre d'altres.



## 2.3 CONCLUSIÓ

Com pot apreciar-se, cada vegada i des de fa anys existeixen sistemes de generació procedimental complexos, complets i que generen resultats realistes. També és cert que els més realistes venen definits per una gran quantitat de paràmetres, mentre que els més senzills d'utilitzar són massa homogenis.

No obstant això, s'està avançant cap a punts ideals on objectes, plantes o mapes poden ser generats per ordinador amb una intervenció humana mínima, alleugerint els costos de disseny gràfic.

El repte que es planteja en aquest projecte és realitzar un mètode senzill que generi models complexos sense necessitat de definir un número elevat de paràmetres.

## 3 ANÀLISI

---

El tipus d'illa que es generarà es veurà definida per les següents característiques:

- **Costes:** Es pot escollir la forma bàsica de les costes operant cercles i quadrats.
- **Distinció entre mar i terreny:** Es parteix d'un pla d'alçada zero i tot allò que sigui més alt que zero serà considerat terreny i el nivell zero o inferior serà considerat aigua.

**Distribució de biomes:** Distribuïts per alçada. Els biomes es generaran per alçades amb cert soroll, començant per sorra/platja i acabant per cims glaçats als punts més alts.

## 4 DISSENY

---

Per a generar terreny s'ha decidit seguir una estratègia basada en la unió de blobs en un mapa d'alçades que, en començar el procés, s'inicialitza a zero sent aquesta l'altura del nivell del mar.

Un blob es una funció que determina les alçades en una àrea al voltat d'un punt i es veuen definits per les següents característiques:

- **Centre:** coordenades en x i y que determinen el centre en el mapa d'alçades. S'ha obligat, a l'algoritme, a generar centres dintre d'un rang per a que cap frontera arribi mai a sortir del HeightMap.
- **Forma:** en aquest cas, pot ser quadrat o rodó. Però bé es podrien introduir certes formes via imatge, com taques, per a generar patrons més estranys.

- Radi/Costat: com que els blobs generats per l'algoritme son rodons o quadrats, aquest valor determina el radi o el costat d'aquestes figures. Ajuden a determinar quines cel·les del heightmap es veuen afectades per el blob.
- Perfil: La funció que determina la silueta d'aquest blob. Pot ser:
  - Semiesfera: crea un cúpula de amb el radi abans mencionat totalment semiesfèrica
  - Cub: alça tots els punts afectats per aquest blob a un mig del radi/costat. Tècnicament es prismàtica, ja que quan la base es quadrada genera un prisma rectangular i quan es rodona un cilindre.
  - Campana de Gauss: respon a la formula  $y = h_{max} \cdot e^{-\left(\frac{x}{\sigma}\right)^2}$  sent  $h_{max}$  el radi/costat,  $x$  la distancia al centre del blob i  $\sigma = 10.0f$  de forma experimental. Amb valors mes alts s'obté un mapa massa pla i amb valors mes petits massa poca costa.

Inicialment, la idea era treballar amb perfils semiesfèrics i anar transformant-los amb sorolls i demés accions però mitjançant l'experimentació s'ha vist que no era possible fer-ho d'una forma no massa complexa pel que s'ha optat per les múltiples representacions aquí mostrades.

El mètode proposat en aquest projecte té les següents etapes:

#### **4.1.1 Generació de punts aleatoris, que seran els centres dels blobs i generació dels blobs a cadascun del punts. Unió dels blobs**

Primer, mitjançant el sistema de generació de enters aleatoris del propi Unity es generen els centres. Aquests es veuen limitats a l'àrea central de mapa d'alçades, sempre al mínim d'un 25% de distancia del marge. En aquest pas també son generades la resta de dades importants del blob: els radis mai mes grans que un 25% de la resolució del mapa d'alçades per, en conjunt amb la limitació a l'hora de generar els centres, evitar creuar els límits. El perfil ve donar per l'usuari i la forma de la base es decidida de forma aleatòria.

A partir d'aquest punt es generen les alçades de cada punt al mapa d'alçades recorrent aquesta matriu i a cada punt es comprova quins sons els blobs que pels que es afectat. Les alçades de tots aquests blobs, trobades amb l'alçada, distancia i formula corresponents, es sumen a cada punt. D'aquesta manera tots els blobs es veuen fusionats a la força i no es necessari fer cap altre pas per assegurar-se que tot queda cohesionat.

#### **4.1.2 Generació de soroll al contorn base i a les alçades**

Si bé amb la suma d'alçades ja s'aconsegueix una petita quota de soroll, aquesta es molt subtil i en mapes amb molt pocs blobs es tot massa regular. Per això s'ha optat per a aplicar soroll de forma manual a tot el terreny.

Els mapes d'alçades de Unity es veuen limitats pel fet que són dades percentuals, valors de 0 a 1, el que poden contenir. Tenint això en compte cal que el sistema d'aplicació de soroll no comprometi aquesta característica.

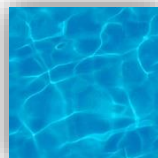
S'opta per fer servir el soroll de Perlin, ja que és el més utilitzat a l'hora de desenvolupar videojocs, i el tenim disponible a la llibreria matemàtica de Unity. Aquest mètode només demana unes coordenades, que en es fan coincidir amb les del punt a tractar al mapa d'alçades, i retorna l'alçada de la funció del soroll. Aquesta alçada, de la mateixa manera que els valors del mapa d'alçades, també és percentual. Donades totes aquestes característiques i tenint en compte el fet de que tot valor igual o inferior a zero al mapa d'alçades no és terreny, sinó aigua, el que ens queda fer és multiplicar els valors retornats per els valors que ja hi ha al mapa. Aquesta forma d'aplicar el soroll redueix una mica de costa, per els valors propers a zero que poden aparèixer, però també la fa una mica més irregular, generant així coses una mica més interessants.

### 4.1.3 Detecció dels biomes

Acte seguit es necessari trobar els biomes adequats. Partim de l'assignació percentual d'alçades següent obtingudes de forma arbitrària:

**Aigua**

**0%**



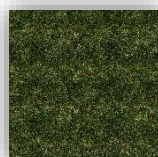
**Platja**

**$\leq 15\%$**



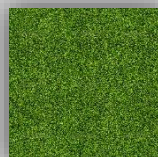
**Vegetació espessa**

**$\leq 50\%$**



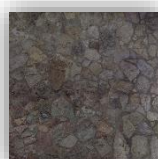
**Vegetació Lleugera**

**$\leq 75\%$**



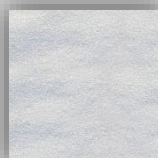
**Roca**

**$\leq 95\%$**



**Neu**

**$< 100\%$**



A partir d'aquí es important saber que això només es pot fer amb una mesh, ja que per a diferenciar biomes es vol aplicar textures, i la forma més efectiva és sobre aquestes. El sistema de color de Terrain és bastant més complex, i desviava el projecte del seu objectiu principal. Però l'assignació de textures es tractarà en el proper apartat.

Per a detectar els biomes, la única cosa que ens falta és trobar els baricentres de cada triangle generat a aquesta Mesh, durant el procés de conversió des de Terrain és la forma més òptima, i llistar aquests triangles segons el que indica la taula mostrada més amunt.

Per a assignar aquests biomes, materials, és necessari fer una llista de triangles als quals se'ls vol assignar cada una de les textures. Per a tal efecte s'ha obtingut un algoritme que converteix els Terrain a Mesh [mem6]. En aquest algoritme es generen vèrtexs, uv i triangles del model així que s'ha aprofitat també per a assignar les llistes dels biomes.

Per a tal efecte primer es localitza l'alçada màxima d'entre tots els vèrtexs per a tenir un valor de referència com a 100%. Acte seguit, a cada triangle trobat, es calcula el baricentre de forma senzilla sent aquest la mitja de les alçades dels punts i assignant-los a la llista que els correspon de triangles seguint la taula donada anteriorment.

#### **4.1.4 Aplicació de les textures i il·luminació final**

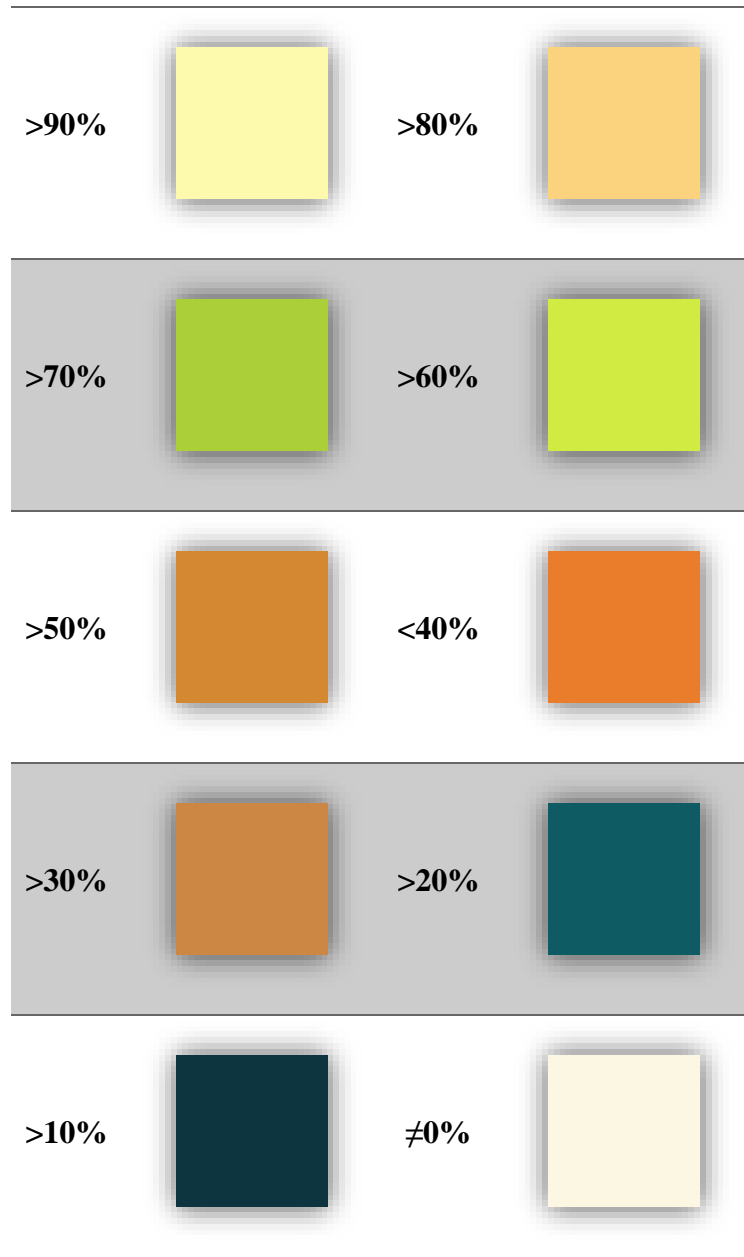
Un cop es té la forma de detectar què pertany a cada bioma és necessari mostrar-ho al usuari mitjançant texturització. La forma més òptima, amb una sola Mesh, és introduint les submeshs.

Una Mesh està definida per triangles, uvs i vèrtexs de forma genèrica. Però l'objecte a Unity inclou altres dades que ens resulten útils com la llista de Material i de Submeshs. Una Submesh és una llista de triangles a la qual se li assignen unes característiques comunes com pot ser un shader o un material diferents a la resta de l'objecte.

Quan es seleccionen quins triangle formen part de cada bioma s'han anat guardant aquests en llistes separades, són arrays d'enters, que un cop acabades es guarden en un ordre en concret dintre de la llista de Submeshs.

Finalment és necessari també tenir creats els materials amb les textures vistes a la taula i assignar-ho tot junt al mesh. Cada un dels materials s'assignarà a cada una de les llistes de triangles, anomenades submeshs, en relació a la seva posició al mesh; primer submesh amb el primer material i així successivament.

Durant el procés de transformació també s'ha inclòs una visualització especial imitant les formes del videojoc independent Godus. Aquest sistema es basa en la subdivisió del model en corbes de nivell i l'assignació de colors plans a cada una d'aquestes corbes, donant així un aspecte més toon a l'estructura creada. Per a recrear aquestes corbes recorrem la llista completa de vèrtexs i aproximem cada posició vertical, deixant intactes les variables horitzontals dels punts, al valor percentual mòdul 10 mes proper.





I tots els valors que son igual a zero conservar la textura d'aigua per defecte ja mostrada en la taula anterior.

En quant a la il·luminació s'ha decidit deixar les llums per defecte, desplaçant-les i incrementant els seu valor per a que faci de sol. Com que Unity no te sistema de llum global fent servir un focus de llum direccional que il·lumina des de dalt amb el color i la intensitat adequats es pot arribar a simular.

## **4.2 OPTIMITZACIÓ:**

Durant el desenvolupament de l'aplicació s'han trobat alguns problemes amb el temps d'execució. Inicialment, el temps de generació de models i d'assignació els triangles als biomes corresponents era massa alt, superant els 2 minuts en alguns casos.

Això es degut a que aquest procés es feia de forma independent al sistema de generació de models, que ja recorre tota la llista de vèrtexs i triangles ja que la crea. Passat un temps de proves i anàlisi cap de les solucions possibles mostra cap tipus de millora en el temps. Es un *for* de 32mil enters en el qual no es fa cap tasca intensiva i a més a més l'esforç es lineal, no varia a cada iteració.

La solució es troba en integrar la tasca de generació de submesh amb el sistema de creació de la pròpia mesh, baixant així el temps d'execució de 2 minuts a dècimes de segon, com es veurà en les proves mes endavant

## **4.3 INTERFÍCIE D'USUARI**

Per finalitzar, cal considerar que aquest programa ha de poder ser utilitzat, i sense una interfície senzilla d'usuari no es viable.

Unity proporciona la classe UI per treballar fàcilment des del *inspector* de l'editor. Per a fer-la servir, cal afegir un GameObject Canvas, assignat a la capa UI ja proporcionada pel sistema i que el RenderMode sigui ScreenSpace, i que com a fills tingui un GameObject en blanc al que s'afegeix un script de control. Script conté tots els mètodes a ser cridats per botons i altres inputs.

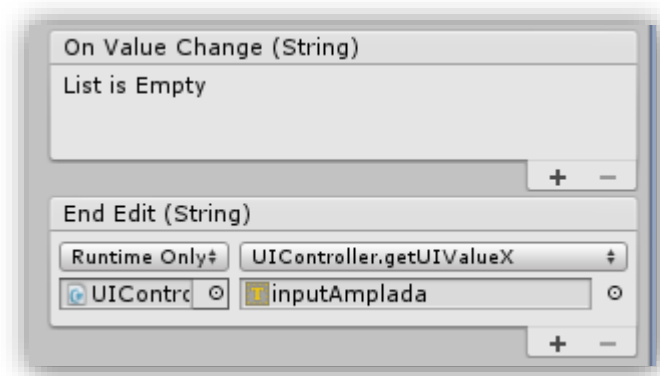
### 4.3.1 Posicionament

Primer s'ha de col·locar cada un dels objectes al seu lloc. Cada un dels items es limita per les següents característiques de posicionament:

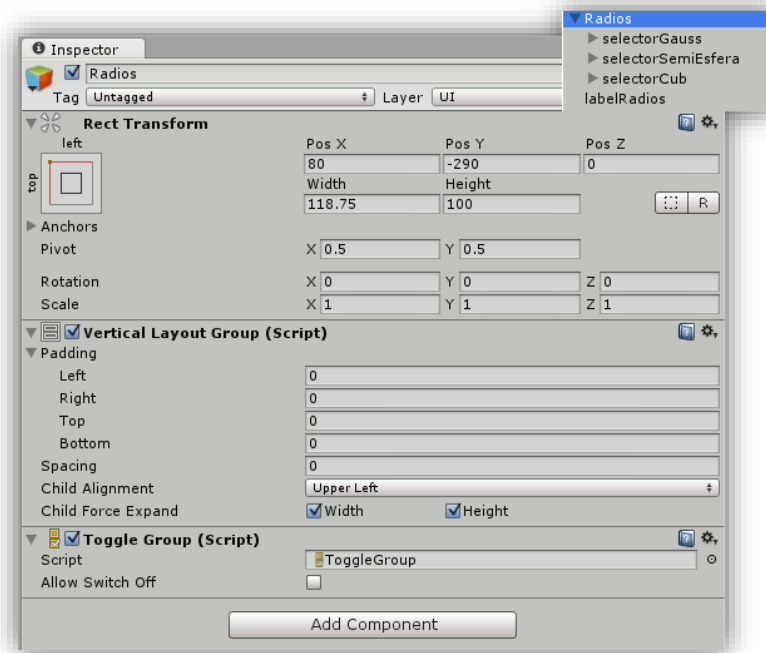
- Punt d'ancoratge: punt considerat l'origen de coordenades de la pantalla per al item en qüestió. Això facilita molt el posicionament relatiu per a múltiples resolucions.
- Coordenades: posició x, y i z respecte al punt d'ancoratge. Com que treballem en un espai bidimensional que es la pantalla la coordenada z és completament irrelevant.

Tots i cada un dels objectes mostrats en pantalla per a us de l'usuari tenen en comú que disposen de listeners independents. Unity, a través de l'editor, proporciona un conjunt de mètodes que s'executen per esdeveniments i així podem tenir control total sobre la presa de dades.

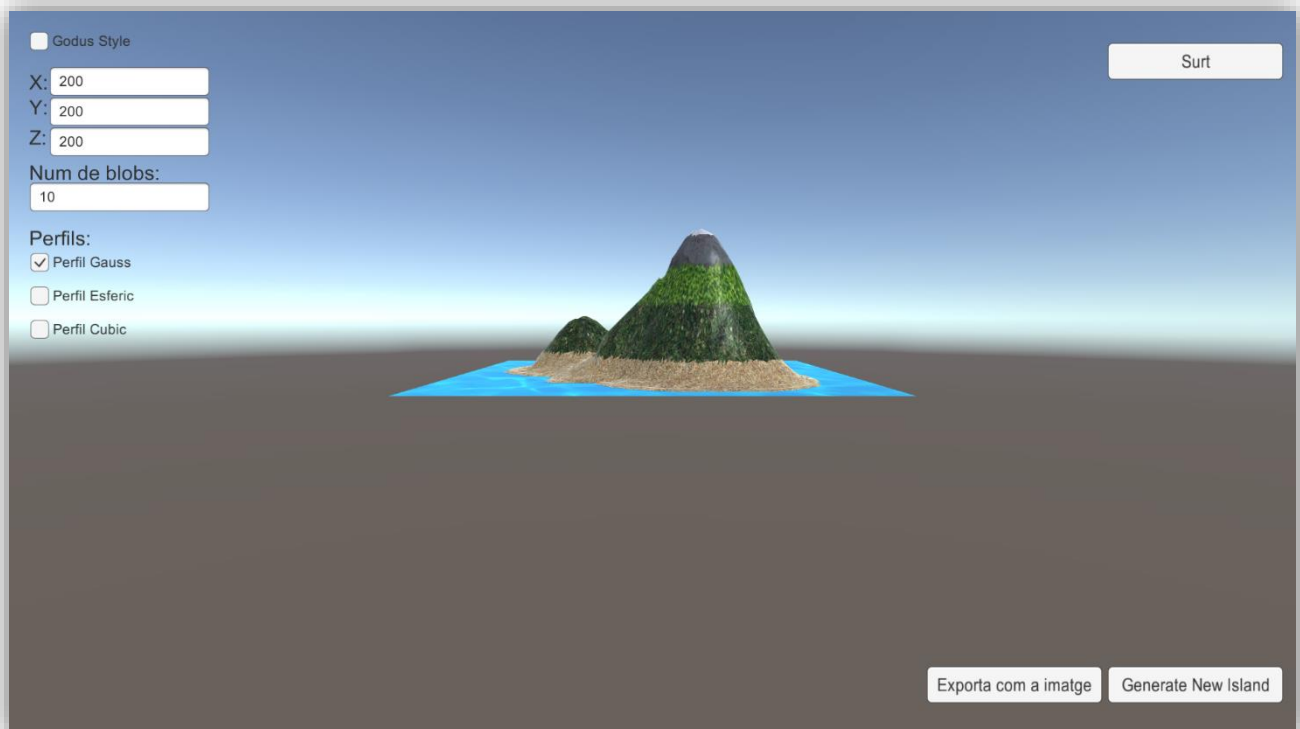
Com que accedir a algunes de les dades d'aquests controls via codi pot ser una mica lent o complex s'opta per a fer servir els mètodes com `OnValueChanged` o `EndEdit` per a modificar variables del controlador, script comú entre tots els objectes en pantalla.



Unity no implementa cap sistema de RadioButtons de forma directa o explícita, però si que ens permet simular-ho via layouts de botons. Per a tal exemple només cal que creem un gameobject en blanc i li afegim el layout de ToggleGroup i a aquest GameObject li afegim com a fills tots els Toggles que vulguem transformar en RadioButtons.

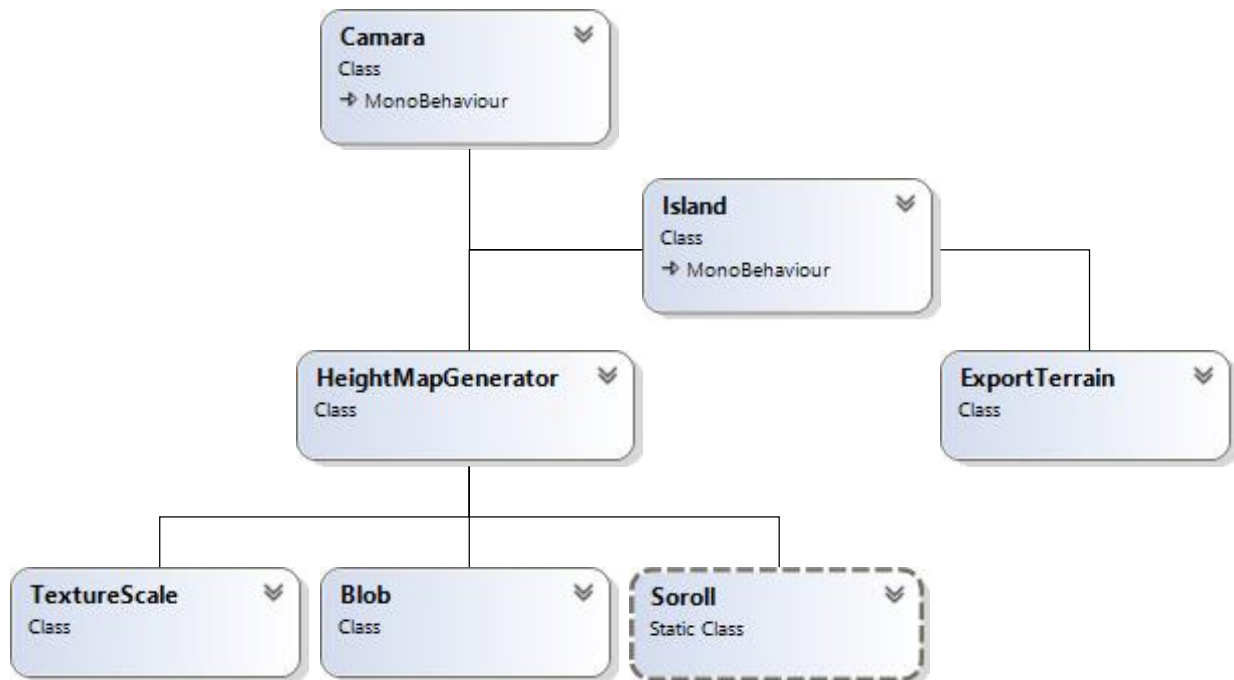


Finalment el resultat d'aquesta interfície es el següent:



## 4.4 DIAGRAMES DE CLASSES

A continuació es detalla un diagrama de classes que suporta tots els mètodes explicats en els apartats anteriors.



Com es pot apreciar, no hi ha herències ni res realment complex al disseny ja que no ha sigut necessari. Tot són relacions d'ús que ens resulten molt útils. Però aquesta només es una part. Donades les característiques de Unity es podria dir que hi ha programes desenvolupats a aquest projecte.

El programa principal es forma de *GameObjects* que contenen *scripts* o classes pròpies de Unity i que s'executen de forma totalment independent entre ells. Per començar, el diagrama mostrat mes amunt indica el funcionament, aproximat, del *GameObject* Generador.

Però a aquesta escena s'han afegit dos mes per a poder manipular-ho tot correctament. Aquests són *Camara* i *UIController*. Aquest últim necessita accedir a *Island* per a poder cridar la generació de les illes, però *Camera* es totalment independent.

## 4.5 CLASSES

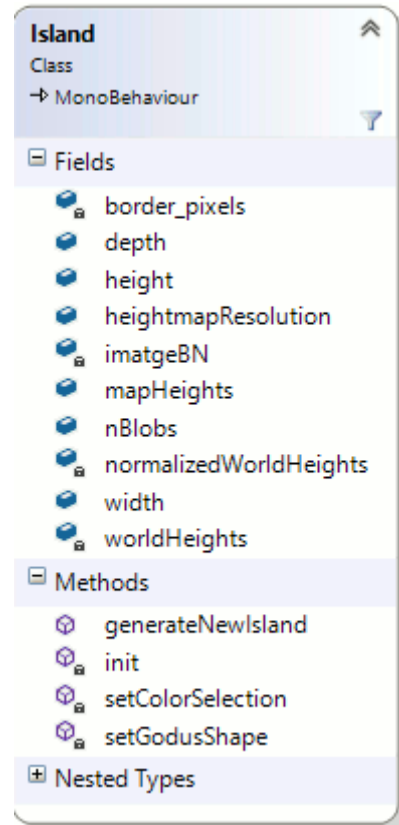
### 4.5.1 Island.cs

Classe principal. S'encarrega de cridar tots els mètodes de la resta de classes i conté totes les dades importants. Es podria dir que és el *main()* de tot el nostre programa.

Podem ressaltar els següents atributs:

- *depth/height/width(int)*: Mida de la illa a generar en les unitats de mesura del món que proporciona Unity.
- *mapHeights(float[[[[]]])*: Mapa d'alçades fet servir al *Terrain*.
- *nBlobs(int)*: nombre de Blobs a generar.

Finalment, només podem ressaltar-hi els mètodes *init()* que ens inicialitza les variables que fem servir i *generateNewIsland()* que arrenca tot el procés de generació d'illes.

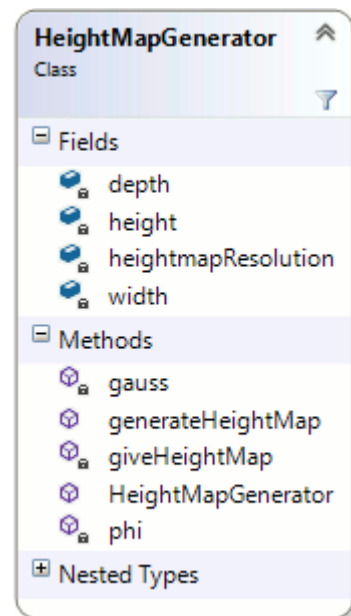


### 4.5.2 HeightmapGenerator.cs

Aquesta segona classe es la primera que es crida en moment que necessitem generar un *HeightMap*. Segueix el procés explicat a l'apartat anterior per a generar els *Blobs* i retornar una imatge correcta per a generar un terreny.

Ressaltem els següents atributs i mètodes:

- *depth/height/width(int)*: Coordenades en tres dimensions del model definitiu. Ens serveixen per inicialitzar el mapa d'alçades en coordenades de món i normalitzar.
- *heightmapResolution(int)*: Defineix l'amplada i alçada del mapa d'alçades definitiu.

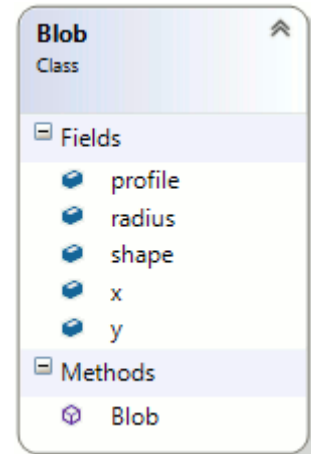


- *generateHeightMap()*: conté tota la lògica de generació de mapes d'alçades. Crida a la resta de mètodes de la classe per a tal d'assolir aquest objectiu i retorna una matriu quadrada de dues dimensions amb el mapa.
- *gauss()/phi()*: càlculs matemàtics necessaris per a generar algunes de les alçades al mapa. Responen a les equacions de la Campana de Gauss.

#### 4.5.3 Blob.cs

Simple classe a nivell de Model. Permet emmagatzemar totes les dades necessàries per a definir un blob i, un cop al generador, fer-lo servir. Donat que no té mètodes mes enllà del constructor es passa a enumerar els seus atributs:

- *x/y(int)*: coordenades del centre d'aquest blob
- *radius(float)*: costat/radi de la figura d'aquest blob
- *shape(int)*: forma de la base del blob. Si és un 0 la considerem circular, altrament, si és un 1, la considerem quadrada
- *profile(int)*: determina la forma del blob per les tres que s'han definit. 0 = gauss, 1 = quadrat, 2 = semiesfera.



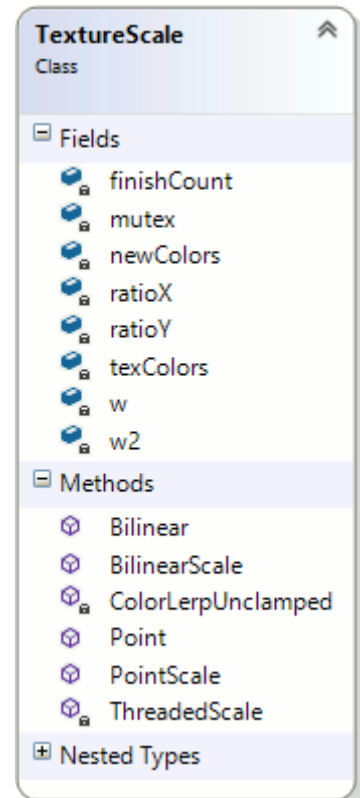
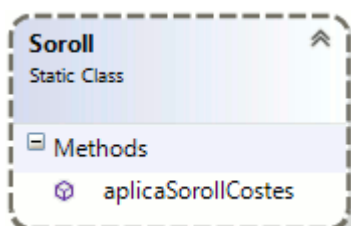
#### 4.5.4 TextureScale.cs

Donat que aquest mètode prové de la comunitat de Unity no s'expliquen al detall els atributs ni mètodes. El *script* retorna una imatge reescalada amb un escalat bilinear amb les mides que se li passen al mètode *BilinearScale()*.

#### 4.5.5 Soroll.cs

Classe estàtica que consta d'un sol mètode. Aquest mètode aplica soroll de Perlin a una imatge donada i la retorna amb aquesta pertorbació. Es fan servir les coordenades de cada píxel per a obtenir el valor del mètode de Perlin proporcionat per les llibreries matemàtiques de Unity i

aquest es multiplica directament al valor del píxel.



#### 4.5.6 ExportTerrain.cs

Aquest mètode també prové, en gran mesura, de les pàgines de la comunitat de Unity però a diferència de *TextureScale.cs* ha sigut modificat per a respondre a les nostres necessitats.

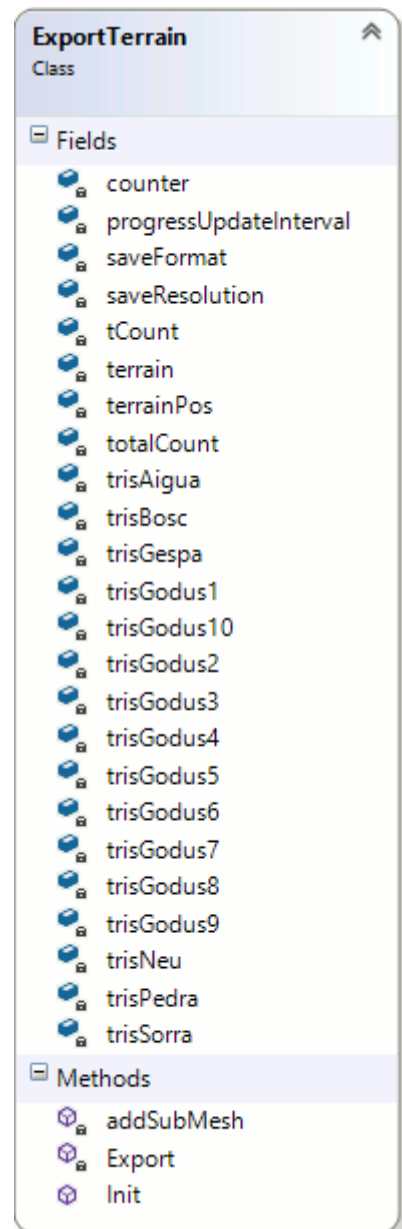
Aquest *script* ens permet transformar un *Terrain* de Unity a Mesh. Originalment exporta els terrenys a arxius \*.obj però s'ha modificat prou com per emmagatzemar aquestes dades en una mesh, assignar-hi submeshs per a texturitzar correctament i retornar-ho al sistema.

Per a tal efecte s'han necessitat els següents atributs:

- *tris\*(int[])*: múltiples variables vector que fem servir per emmagatzemar els triangles generats.  
\* fa referència a tots els atributs que comencen per la paraula *tris* i continuen per qualsevol altre cosa.

La resta d'atributs son usats per l'algoritme original.

Al mètode *export()* és on s'han fet totes les modificacions. Com s'ha explicat abans, aquest mètode genera vèrtexs, *uvs*, i triangles. S'ha afegit una modificació per a que a més a més es generin *submeshs* segons les necessitats de l'usuari i es volqui tot a un *mesh*.



#### 4.5.7 Camara.cs

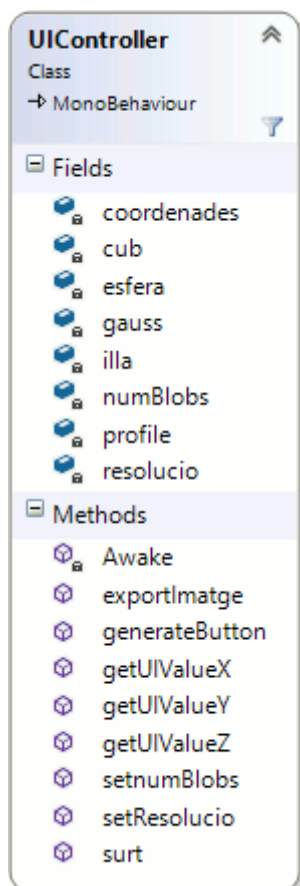
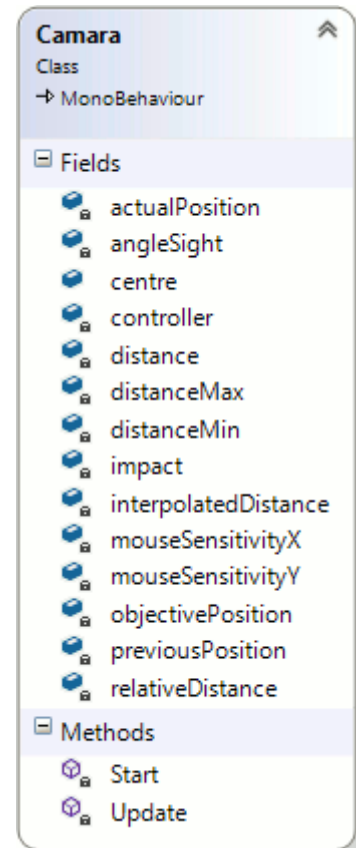
Controlador de la càmera. Al mètode Update es capturen tots els inputs i s'actua en consideració. Fa un us intens de la llibreria matemàtica de Unity i, mitjançant un conjunt d'operacions, implementa un moviment de càmera suau i un zoom precís.

També es té en consideració la posició del objecte *Camera* respecte la illa per a detectar col·lisions i així evitar travessar el model que s'està visualitzant.

#### 4.5.8 UIController.cs

Aquesta classe esta implementada, igual que *Camara*, de forma independent al sistema principal.

Te com a objectiu generar i controlar la interfície d'usuari, generada de forma manual mitjançant el Inspector del motor de joc.



Atributs:

- *Illa(Island)*: referencia a la illa original per a poder cridar el mètode de generació de mapes
- *numBlobs/cub/esfera(int)*: dades que es capturen al vol un cop son editades per a poder introduir-les al generador quan sigui necessari.

Mètodes:

- *exportImage()(void)*: sistema d'exportació d'imatges per a desar el mapa d'alçades actualment visible. Volca sempre la imatge al mateix fitxer ja que Unity no té un mètode senzill per a invocar un diàleg de desat de fitxers.
- *getUIValue\*()/ setNumBlobs()/ setResolució()(void)*: mètodes cridats mitjançant els esdeveniments dels botons i d'altres controladors per a desar les dades corresponents als seus atributs.

- *generateButton()(void)*: crida a la generació d'illes

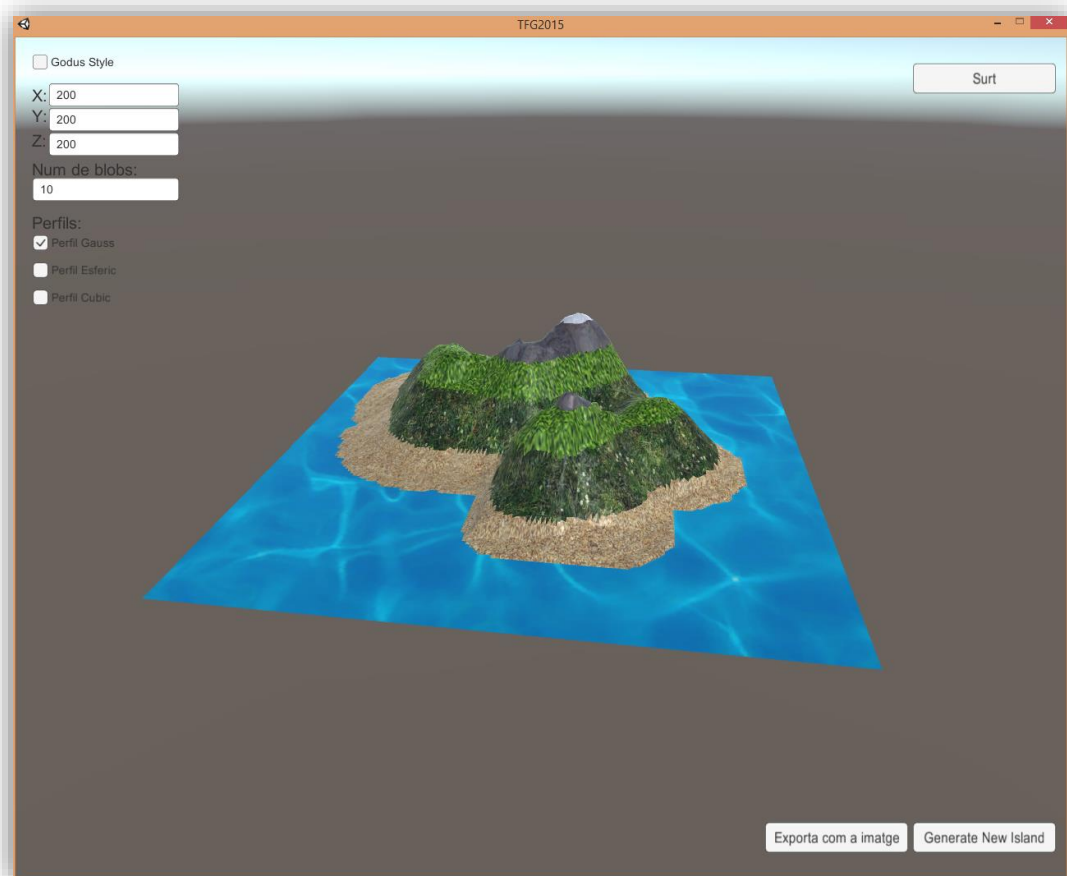


## 5 RESULTATS I SIMULACIONS

---

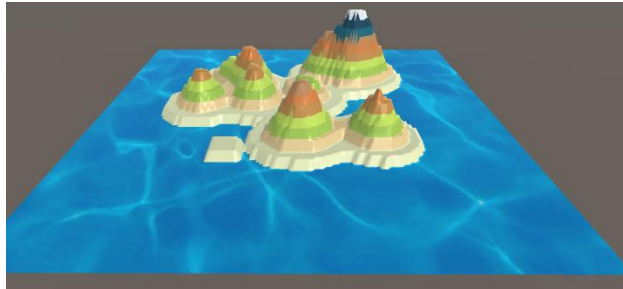
### 5.1 FUNCIONAMENT DE L'APLICACIÓ

A continuació es detalla com funciona l'aplicació. Només començar, es demana la resolució de pantalla en la que es treballarà i tot seguit ja es poden començar a generar illes posant els valors en els paràgraf de la interfície



Aquesta es la finestra inicial que es visualitza al executar el programa. Com es pot apreciar es veuen totes les característiques que podem modificar i que es passen a enumerar a continuació

- 1- Toggle Godus: permet transformar la propera generació a un estil gràfic diferent, per capes. L'alçada serà dividida en 10 parts i a cada una li correspondrà un color diferent, donant-li al conjunt un efecte mes còmic.



☐ Godus Style
 1

X: 200
 2

Y: 200

Z: 200

Num de blobs:
 10
 3

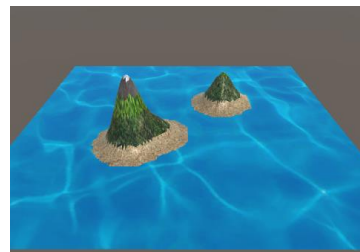
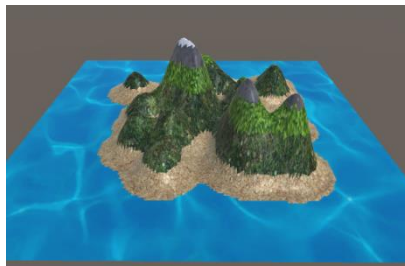
Perfils:
 4

☒ Perfil Gauss

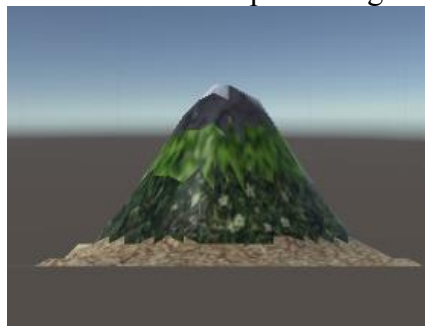
☐ Perfil Esferic

☐ Perfil Cubic

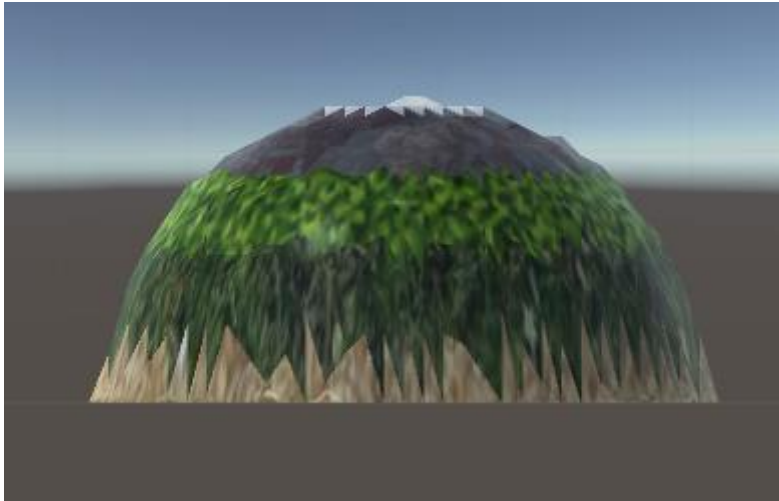
- 2- Selector de mida: altera l'alçada, amplada i profunditat del model generat en unitats de mon, sent Y l'alçada i X i Z l'amplada i la profunditat respectivament.
- 3- Numero de blobs: la quantitat de protuberàncies creades per a generar l'escenari. Un excés d'aquestes pot portar a un mapa sobresaturat o a temps de carrega massa grans i, amb això, un bloqueig temporal de l'aplicació. Es mostra un exemple de 50 blobs vs 5 blobs.



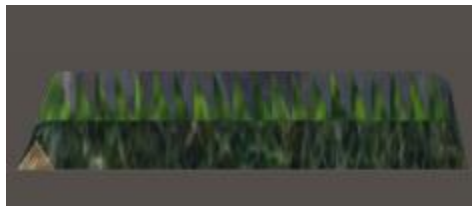
- 4- Perfils: la forma del perfil d'un sol blob
  - a. Gauss: el blob agafarà la forma d'una campana de Gauss, forma relativament natural i amb mes percentatge de sorra.



- b. Esfèric: cada blob prendrà una base totalment rodona i un perfil de semiesfera.



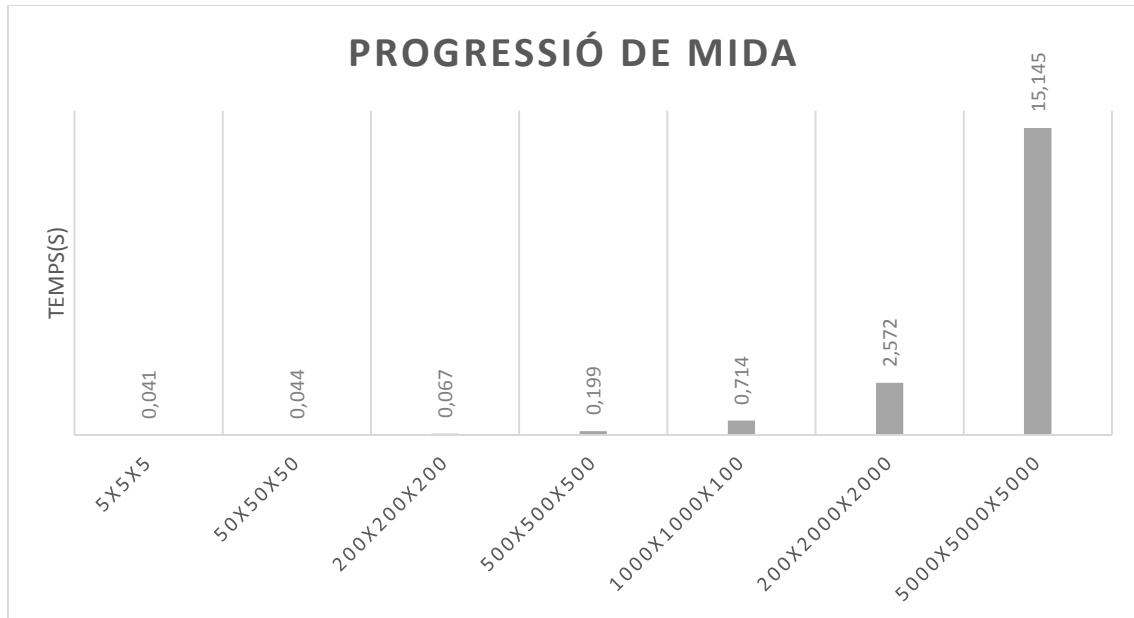
- c. Cub: tots els blobs s'alcen de forma totalment vertical un terç del seu diàmetre. En aquest cas es quan es mes visible el fet de que, de forma aleatòria, tinc blobs amb la base quadrada o rodona.



## 5.2 PROVES DE RENDIMENT

### 5.2.1 Incrementant la mida del mapa

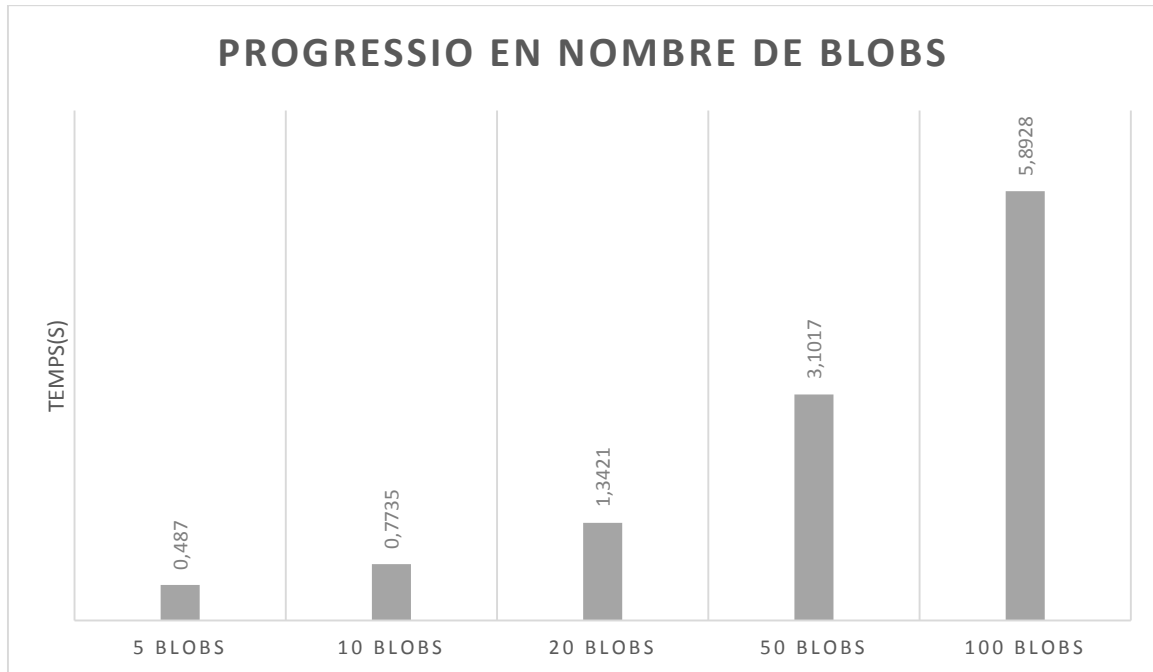
A continuació s'exploren les proves realitzades per avaluar el rendiment en temps quan s'incrementen les mides del mapa.



Cada mesura ha estat presa 10 vegades i aquí es mostra la mitjana d'aquestes 10 tot amb 10 blobs a cada una. El temps esta indicat en segons. Com podem veure, a mida que creix l'àrea a treballar el consum puja exponencialment. També cal tenir en compte que a la primera i les dues últimes mides el mapa era completament impossible de visualitzar. A la 5x5x5 perquè tot sol ja era una cel·la i en els últims perquè la distància de dibuixat es mes curta.

### 5.2.2 Incrementant el numero de blobs

Acte seguit es fa una prova de la progressió del temps d'execució de l'algorisme de generació de mapes d'alçada a mida que s'incrementa el nombre de blobs. S'exagera la mida del mapa per assegurar lectures amb variacions significatives i perceptibles.















De la mateixa manera que abans el temps es en segons i el resultat es la mitja de 10 mesures. Tot això ha estat mesurat amb un mapa de 1000x1000x1000 per a que, en cas de que hi hagi diferències realment grans, fossin més palpables.

## 6 VALORACIÓ ECONÒMICA

### 6.1 ANÀLISI DEL TEMPS DE REALITZACIÓ DEL PROJECTE

Per a aquest projecte s'ha treballat de 4 a 8 hores setmanals, de mitjana, des de principis de febrer. Tenint en compte això i que totes les fases s'ha solapat en algun moment o altres es pot veure el proper diagrama de Gantt. Els apartats d'anàlisi i implementació porten implícits unes fases de testing, imprescindibles per a seguir treballant.

	FEBRER	MARÇ	ABRIL	MAIG	JUNY
Formació					
Anàlisi					
Implementació					
Documentació					

### 6.2 VALORACIÓ DEL COST ECONÒMIC DEL PROJECTE

S'han exclòs del cost econòmic del projecte les hores de formació i, de les hores de documentació, les de la memòria, computant únicament les de documentació del manual d'usuari.

Si es fa cas dels càlculs mostrats al diagrama de Gantt, que son molt aproximats ja que no s'ha fet cap seguiment real ni precís del temps de treball els resultats son els següents:

- Cost anàlisi =  $(6 \times 4) \times 80 = 1920\text{€}$
- Cost programació =  $(15 \times 4) \times 60 = 3600\text{€}$
- Cost documentació =  $3 \times 50 = 150\text{€}$

## 7 CONCLUSIÓ

---

Durant el desenvolupament d'aquest projecte s'han plantejat uns objectius a l'apartat d'Anàlisi. El primer de tots, tenir múltiples siluetes de illa per escollir, s'ha complert clarament. No només hi ha múltiples perfils verticals per escollir sinó que aquests tenen bases diferents i s'hi ha afegit un perfil de fantasia anomenat Godus que ens tradueix el mapa a corbes de nivell.

Acte seguit es plantejava la diferenciació entre aigua i terreny, que al resultat final es molt visible i es una forma feta servir molt habitualment. Finalment, està la selecció de biomes. Altra vegada es veu de forma directa el resultat i que hi ha dos biomes per escollir.

La primera contribució d'aquest projecte és l'estudi dels mètodes procedimentals per a generar automàticament escenes. Aquest estudi ha permès la implementació del programa en el que es basa tot el projecte i l'estudi de les restriccions necessàries per a que aquest pugui ser acabat.

Acte seguit es important recalcar la importància del producte final, originat d'aquest estudi, que es la implementació d'un algorisme de creació d'illes en Unity en comptes de una forma genèrica.

En quant a l'estudi, s'ha realitzat una primera aproximació als mètodes més rellevants de la generació procedimental de tot tipus però incidint expressament en el algorismes de generació de terrenys i, més concretament, al camp de les Illes.

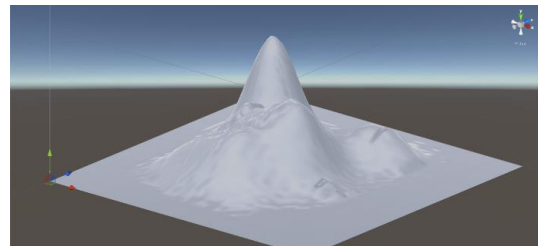
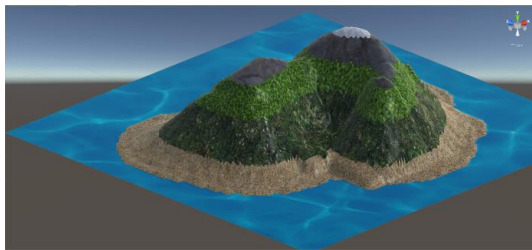
Aquest algoritme, implementat al motor de joc Unity, té moltes possibles millores:

- Aplicar mascarees de soroll a la costa: d'aquesta manera s'alteraria el perfil d'una forma més natural, generant caps i golfs més realistes.
- Plantilles de forma per a blobs de forma que no sigui constant: una implementació del sistema de plantilles pot afegir varietat al sistema.
- Exportació a \*.obj : d'aquesta manera qualsevol dissenyador pot fer servir els models generats a un programa d'edició de models 3D.
- Millores en la generació de coordenades de Blobs: actualment es solapen més blobs dels desitjats, una revisió de l'algorisme milloraria el producte final.
- Conques fluvials: només es genera un model de terreny, però la vegetació i l'aigua

s'ha ignorat completament. La implementació de llacs i rius seria ben rebuda.

- Mapes de vegetació: de la mateixa manera que les conques fluvials, també es pot estudiar l'addició de gespa i models complets de plantes.

Entre d'altres, també entre els extrems aportats, es pot modificar el sistema Godus per a acceptar un nombre variable d'alçades i així fer les corbes de nivell d'alçada constant, en comptes de percentual com es ara. I millorar el sistema de texturitzat per a assegurar uns biomes mes complexos, si es millora prou es pot acabar treballant amb Terrain en detriment del Mesh i així superar el límit de triangles i vèrtexs d'aquests últims.



Com be es pot comprovar en aquesta comparativa, a la dreta Mesh i a l'esquerra Terrain, la definició del terrain es mes gran, però el mesh permet colorar molt mes fàcilment.

Finalment, un dels grans enigmes plantejats al projecte es sobre la coherència de generació. Quin era el procés de generació que assegurava la cohesió del tots els elements a un sistema generat procedimentalment. Em el cas de les illes es pot concloure que el fet de generar perfils realistes i formes de costes creïbles assegura gran part d'aquesta coherència.



## 8 REFERÈNCIES BIBLIOGRÀFIQUES

---

- [mem1] R.M. Smelik et al. Published in COMPUTER GRAPHICS Forum.<http://doi.org/10.1111/cgf.12276>
- [mem2] PEYTAVIE A., GALIN E., GROSJEAN J., MERILLOUS.: Arches: a framework for modeling complex terrains. In Computer Graphics Forum: Proceedings of Eurographics 2009 (2009), Eurographics Association.
- [mem3] FOURNIER A., FUSSELL D., CARPENTER L.: Computer rendering of stochastic models. Communications of the ACM.
- [mem4] ZHOU H., SUN J., TURK G., REHG J. M.: Terrain synthesis from digital elevation models. IEEE Transactions on Visualization and Computer Graphics 13, 4 (July-Aug. 2007).
- [mem5] Poligonal Map Generation for Games, Red Blob Games. <http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/>
- [mem6] Terrain exporter for Unity. <http://wiki.unity3d.com/index.php?title=TerrainObjExporter>
- API de Unity. <http://docs.unity3d.com/ScriptReference/>
- Texture 2D Resize. <http://wiki.unity3d.com/index.php/TextureScale>

## APÈNDIX: MANUAL D'USUARI DE L'APLICACIÓ

---

La màquina amb mínimes característiques amb la que hem pogut fer proves i que ha funcionat correctament ha sigut un equip amb:

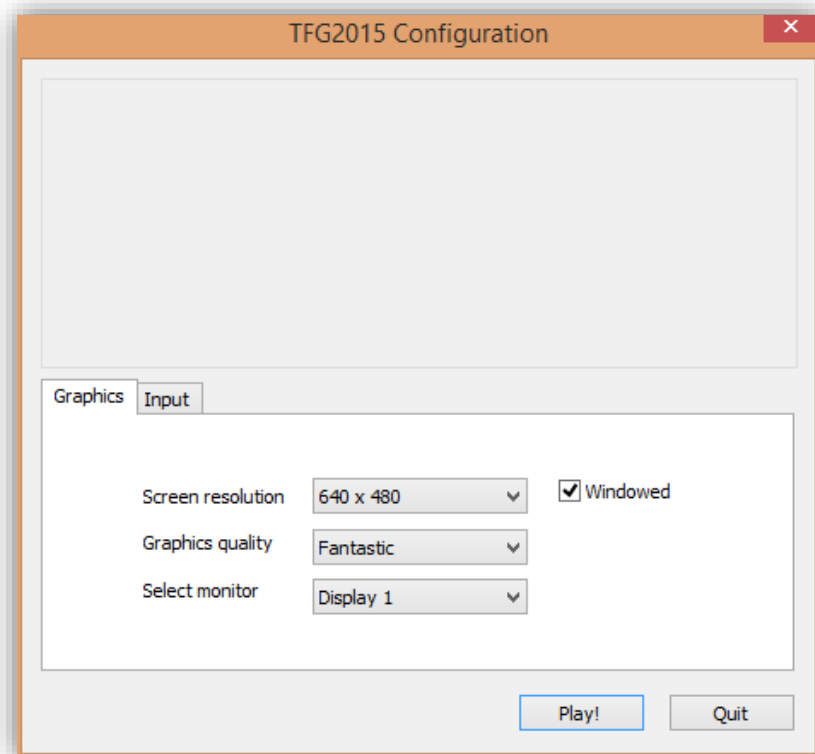
- Processador Intel Core 2 Duo a 2.13GHz
- 1GB RAM
- Windows 7
- Targeta gràfica integrada Intel

Hi ha builds del programa per a Windows de 32 i 64 bits i per a MacOSX de 32 bits. Aquesta última no ha sigut testada ja que no es disposa de cap Mac a prop.

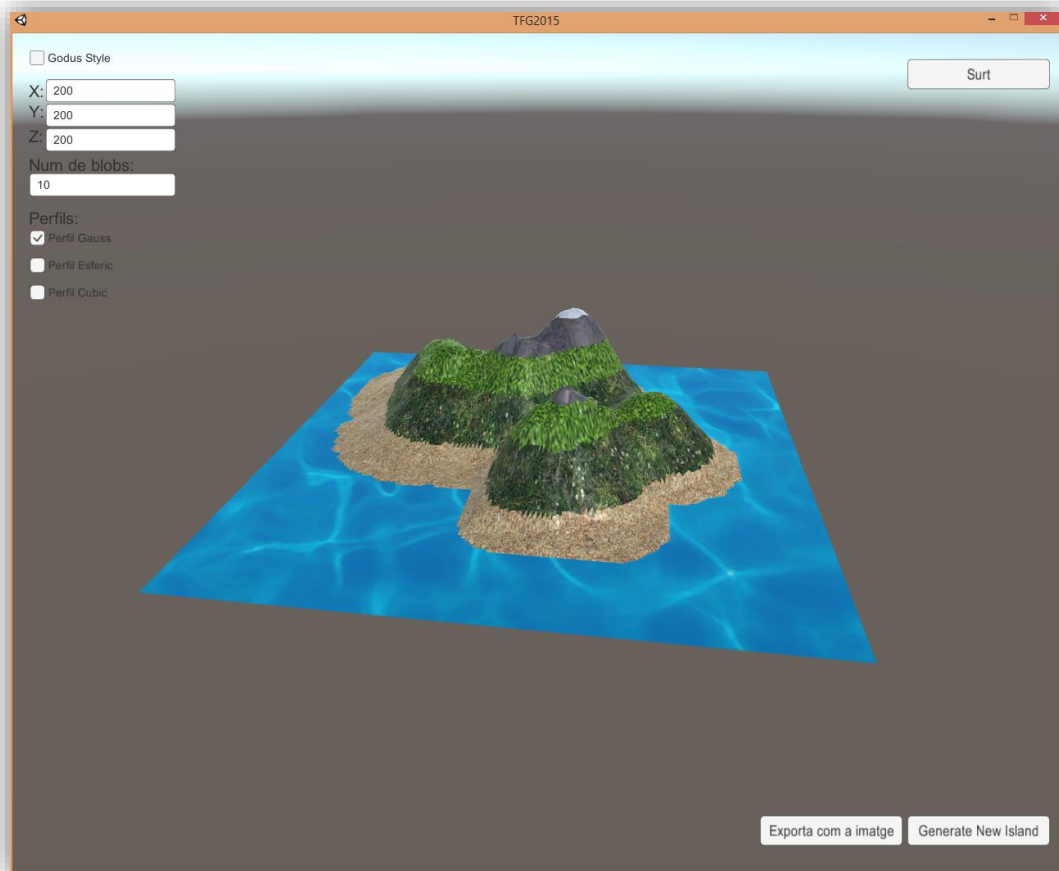
Necessari descomprimir el fitxer *codi.zip* i obrir la carpeta *build*.

Dintre hi trobarem les versions per a les quals esta el programa, Windows de 32 i 64 bits o MacOS. Accedint a la que correspongui podrem veure i podrem executar amb doble clic el fitxer anomenat *TFG*.

Ens donarà la benvinguda la següent finestra:



Es selecciona la resolució que mes ens convingui i el monitor de sortida i es fa clic a *Play!* per a executar el programa.



Aquesta es la finestra principal del programa. Per a controlar la càmera podem fer-ho arrossegant el clic esquerre del ratolí i fer zoom amb la rodeta. Des de aquests controls podem experimentar a l'hora de generar contingut.



1. Modificar el mode visualització de realista a *Godus*
2. Modificar l'alçada, amplada i profunditat del producte final en unitats de mon
3. Nombre de blobs generats
4. Perfil a generar de l'illa

Un cop modificades aquestes dades si es fa clic al botó *Generar nova illa* es descartarà la illa que s'està visualitzant en aquest moment i apareixerà una de nova en qüestió d'instant.

Si es prem *Exporta com a imatge* el mapa d'alçades d'aquesta illa, en resolució de 256x256 es desarà a la mateixa carpeta on s'estigui executant el programa amb el nom *imgFinal.png*. **IMPORTANT:** Aquest botó sobreescrirà el fitxer anterior si no s'ha canviat el nom.

Finalment, per a sortir de l'aplicació cal polsar el boto *Surt* i es tancarà automàticament tot el programa.