SOMesay

Andrea Carrrarini 927539

February 2021

UNIVERSITÀ
DEGLI STUDI
DI MILANO

LA STATALE

Indice

Elenco delle figure

1 Introduzione

Questo progetto, realizzato in Unity3D, mira a realizzare uno strumento di aiuto nella creazione del mondo di gioco per dei videogiochi "open world", analizzando il terreno, generato proceduralmente in precedenza, e marcando delle aree come "suitable" per l'insediamento.
Per fare ciò ho realizzato una "Griglia" a partire da una Self Organizing Map (SOM) fissando i neuroni nelle loro componenti X e Y della loro posizione iniziale e lasciando come unico grado di libertà la coordinata Z del neurone.
Questo, unito ad un algoritmo basato su quello delle SOM e all'uso di un Asset Package per la generazione procedurale di Terrain in Unity3D, ha portato al risultato presentato.
Ho lavorato a questo progetto per il corso di "Sistemi Intelligenti Avanzati" durante l'a.a. 2019/2020.

2 Risultato ottenuto

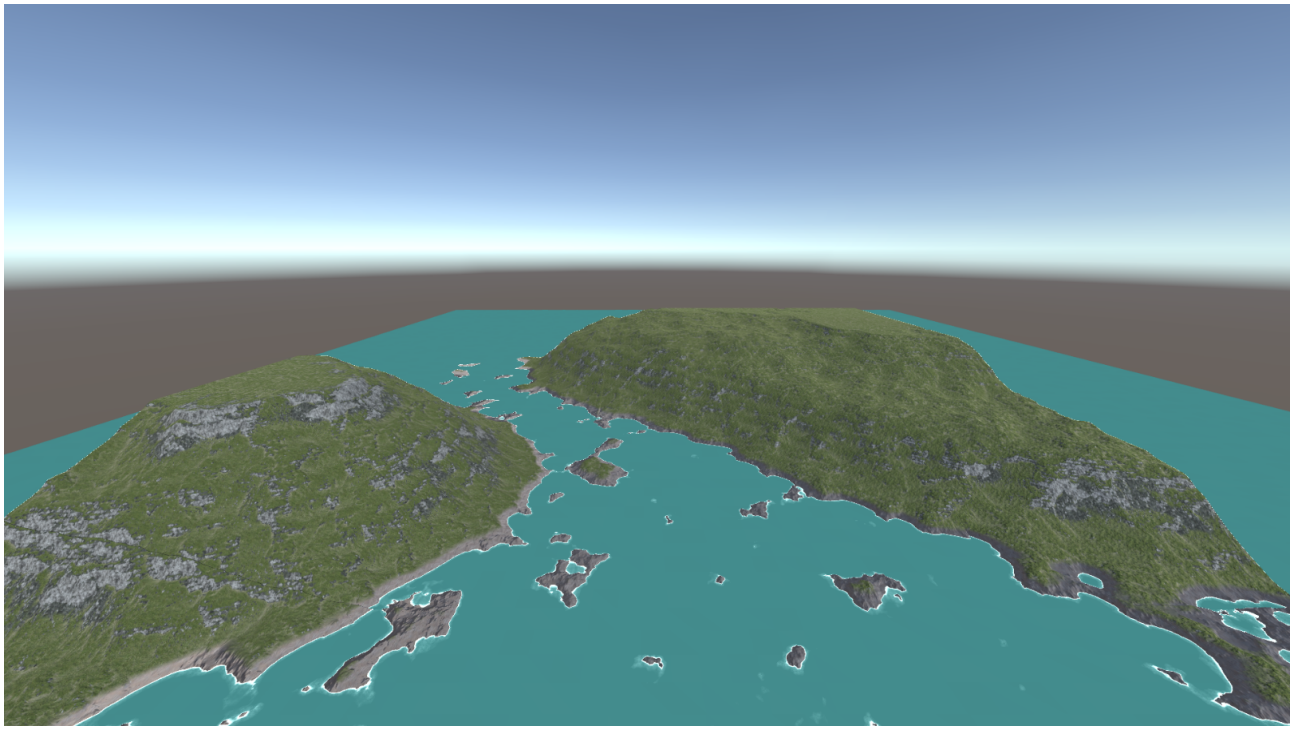

Figura 1: terreno base generato proceduralmente

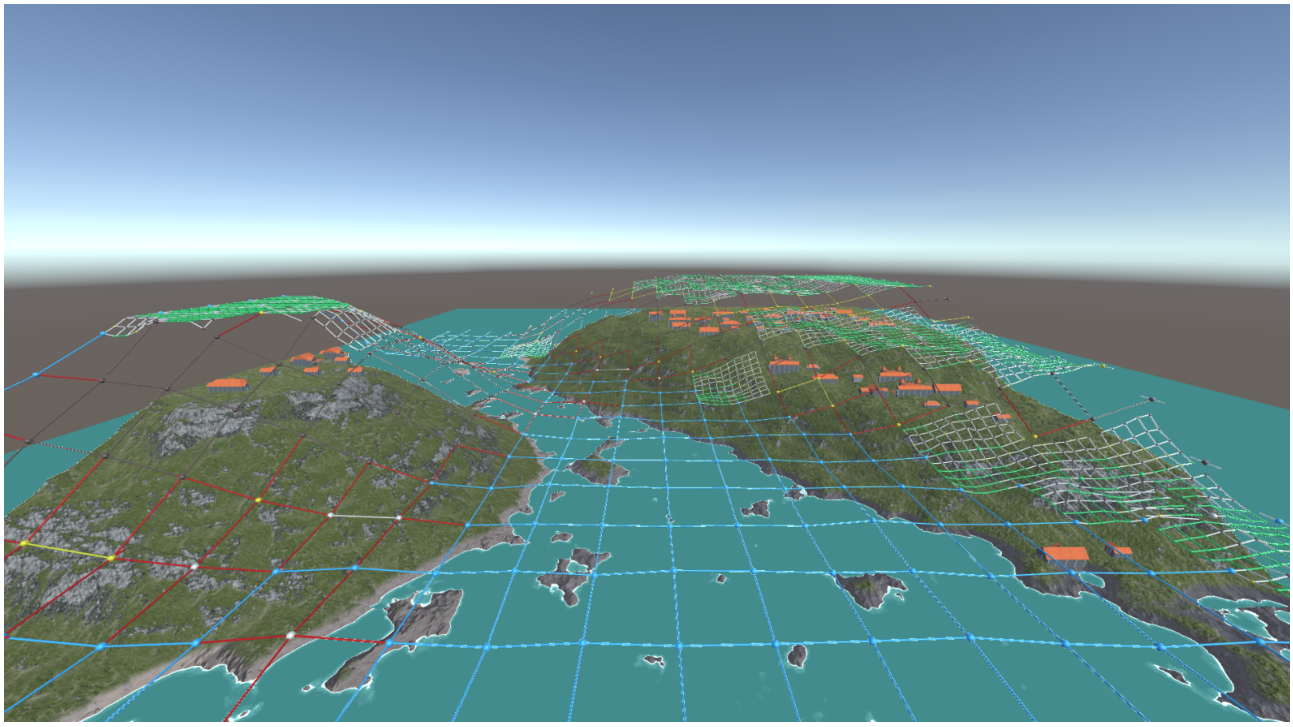

Figura 2: terreno processato con edifici posizionati

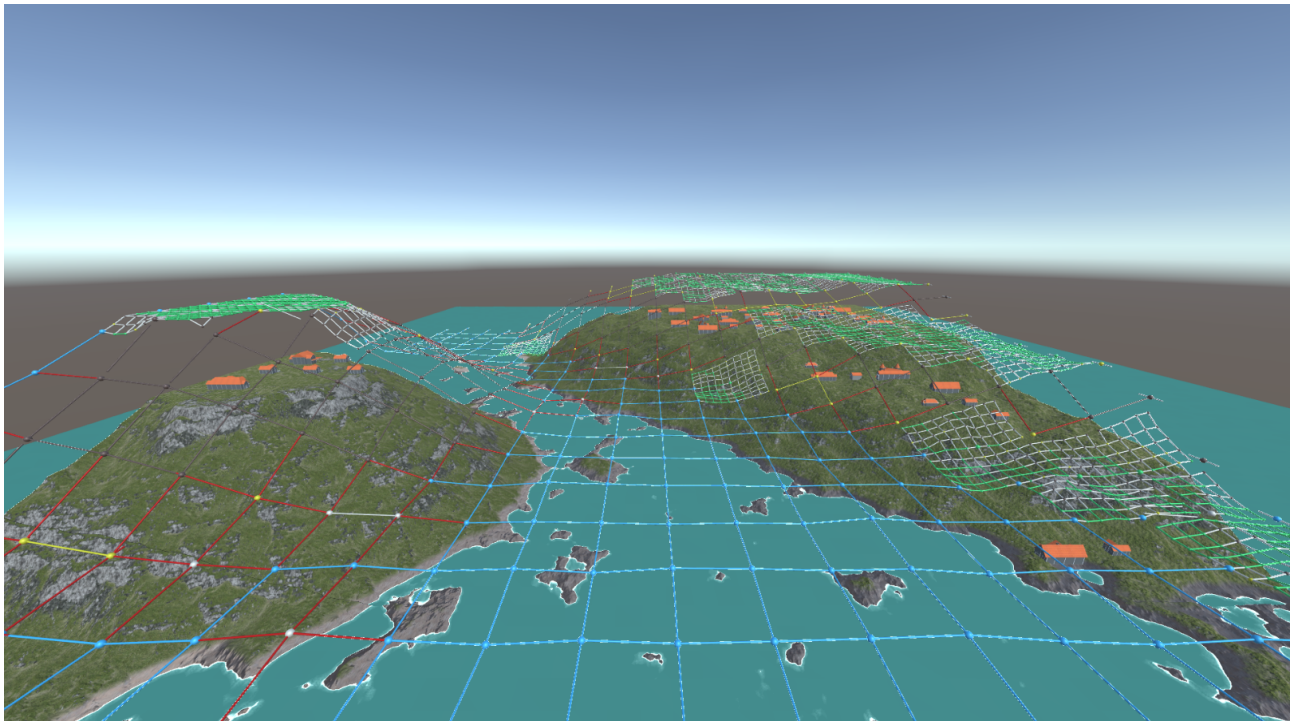

Figura 3: risultato finale dopo la rimozione di alcuni edifici a seguito dell'infittimento della Griglia

3 Implementazione

SOMesay prevede diversi passaggi nella sua esecuzione:

1. il terreno generato proceduralmente viene campionato in m^2 punti organizzati secondo una griglia a cella quadrata regolare e il vettore posizione del punto viene salvato in una matrice mxm (matrice del campionamento).
2. viene preparata la Matrice di Input m^2x3 (m^2 sono i punti campionati e 3 sono le coordinate X, Y e Z).
3. viene inizializzata la Griglia con nxn Neuroni posizionati a coprire completamente il terreno con coordinata Z = 0.
4. la Matrice di Input viene data come input alla Griglia che tramite l'algoritmo delle SOM processa i dati e modifica la coordinata Z dei neuroni della Griglia.
5. al termine dell'algoritmo delle SOM, i neuroni vengono processati e viene assegnato loro un tipo tra i 5 disponibili: SEA, SHORE, PLAIN, HILL e UNSUITABLE.
6. gli edifici vengono posizionati sul terreno in corrispondenza di ogni neurone di tipo HILL.
7. per ogni neurone di tipo HILL vengono analizzate le 4 celle intorno e la griglia viene infittita localmente di nxn per ogni cella coinvolta.
8. i nuovi "sub-neuroni" vengono processati a loro volta e viene loro assegnato un tipo tra i 5 disponibili.
9. per ogni sub-neurone si controlla se l'asse Z passante per la sua posizione interseca un qualsiasi edificio e se sì, quest'ultimo viene rimosso.

3.1 Algoritmo SOM

```
def SOM_Algorithm( input[] ):

    while current_iteration < max_iterations:

        current_radius = calculate_neighborhood_radius( current_iteration )

        BMU = calculate_BMU( current_input )

        learning_rate = calculate_learning_rate( base_learning_rate, current_iteration )

        neurons_in_radius[] = get_neighbors( BMU, current_radius )

        for processing_neuron in neurons_in_radius:

            distance_drop = calculate_distance_drop( processing_neuron, current_radius )

            processing_neuron = update_weights( current_input, distance_drop, learning_rate )

        current_iteration++
```

Figura 4: pseudocodice dell'algoritmo di apprendimento delle SOM

La prima cosa da fare in un'iterazione è calcolare il **Neighborhood Radius** $\sigma(t)$:

$$\sigma(t) = \sigma_0 e^{-\frac{t}{\lambda}}$$

Dove σ_0 rappresenta il raggio della mappa, t l'iterazione corrente e λ la costante temporale definita come:

$$\lambda = \frac{k}{\sigma_0}$$

Con k = numero di iterazioni.

Si procede poi all'elezione del **Best Matching Unit** (BMU) per l'input corrente, ovvero il neurone i cui pesi *Weights(i)* sono più vicini ai valori dell'input corrente. Per fare ciò viene calcolata la **Distance**:

$$Distance = \sqrt{\sum_{i=0}^{n}(input_i - weight_i)^2}$$

Il neurone con la distanza minima dall'input corrente viene eletto come BMU.
In seguito vengono estratti tutti i neuroni all'interno del Neighborhood Radius e per ognuno viene calcolata la **Distance Drop**:

$$\Theta(t) = e^{-(\frac{distBMU}{2\sigma(t)})^2}$$

Dove *distBMU* rappresenta la distanza euclidea tra il neurone processato e il BMU.
Infine si procede all'aggiornamento dei pesi dei neuroni processati:

$$weight(t+1) = weight(t) + \Theta(t)L(t)(input(t) - weight(t))$$

Dove *L(t)* rappresenta il **Learning Rate**:

$$L(t) = L_0 e^{-\frac{t}{\lambda}}$$

Con L_O = *base learning rate.*
Come si può vedere *L(t)* è un'esponenziale negariva, perciò il tasso di apprendimento, e quindi l'aggiornamento dei pesi, decresce con l'aumentare delle iterazioni.

3.2 Significato dei Pesi

Dato che la Griglia utilizzata concede ai suoi neuroni un solo grado di libertà, la coordinata Z, è sufficiente un solo peso per ogni neurone, nel quale ho identificato appunto la coordinata Z del neurone, inizializzato ad un valore randomico.

3.3 Assegnazione dei tipi

```
def Types_Assignment(SOM_map):

    for i in range SOM_map.width:

        for j in range SOM_map.height:

            if SOM_map.get_neuron(i, j).Z <= sea_height:

                SOM_map.get_neuron(i, j).type = SEA

            else if SOM_map.get_neuron(i, j).Z > sea_height and SOM_map.get_neuron(i, j).Z <= shore_height:

                SOM_map.get_neuron(i, j).type = SHORE

            if SOM_map.get_neuron(i, j).type != SEA and SOM_map.get_neuron(i, j).type != SHORE:

                if Math.Abs(SOM_map.get_neuron(i, j).Z - SOM_map.get_neuron(i - 1, j).Z) < plain_slope and
                    Math.Abs(SOM_map.get_neuron(i, j).Z - SOM_map.get_neuron(i, j - 1).Z) < plain_slope:

                    SOM_map.get_neuron(i, j).type = PLAIN

                else if Math.Abs(SOM_map.get_neuron(i, j).Z - SOM_map.get_neuron(i - 1, j).Z) < hill_slope and
                    Math.Abs(SOM_map.get_neuron(i, j).Z - SOM_map.get_neuron(i, j - 1).Z) < hill_slope:

                    SOM_map.get_neuron(i, j).type = HILL

                else:

                    SOM_map.get_neuron(i, j).type = UNSUITABLE
```

Figura 5: pseudocodice dell'algoritmo di assegnazione dei tipi

Per assegnare il tipo corretto a ogni neurone cicliamo su di essi e per ognuno per prima cosa controlliamo se la loro coordinata Z si trovi al dì sotto del livello del mare, nel qual caso il neurone è di tipo SEA, o tra il livello del mare e il limite della spiaggia, eventualità che assegna al neurone il tipo SHORE.

Poi controlliamo che il neurone non sia di tipo SEA o SHORE e in tal caso verifichiamo che lo **Slope**, ovvero la pendenza del terreno, nel nostro caso semplicemente la differenza di coordinata Z in valore assoluto, tra il neurone *N(i,j)* e rispettivamente il neurone *N(i - 1,j)*, il precedente sulla stessa riga, e il neurone *N(i,j - 1)*, il precedente sulla stessa colonna non sia superiore al **plain slope**, limite entro il quale un neurone è definito PLAIN.
Se invece lo slope è compreso tra il plain slope e l'**hill slope** il neurone sarà di tipo HILL.
Nel restante caso il neurone sarà di tipo UNSUITABLE.

3.4 Posizionamento edifici e correzione tramite infittimento della Griglia

A questo punto viene posizionato un edificio in corrispondenza di ogni neurone di tipo PLAIN presente, ma per evitare che un edificio sporga dal limite di un'area di tipo PLAIN verso una di tipo diverso (vedi Figura 6) si procede ad un infittimento della griglia intorno ad ogni neurone di tipo PLAIN.

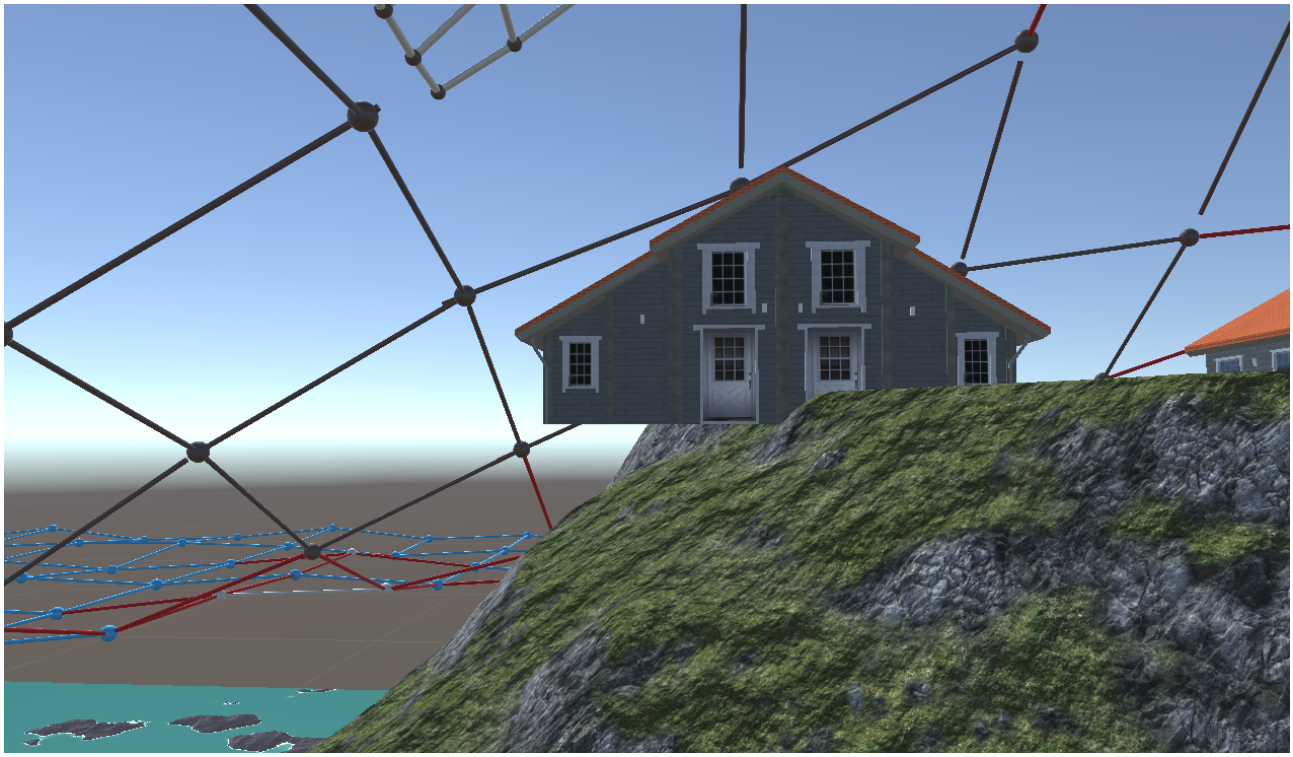

Figura 6: esempio di edificio che sporge

Per fare ciò vengono creati *nxn* "**sub-neuroni**" per ognuna delle 4 celle a formare un quadrato con centro ogni neurone di tipo PLAIN e vengono disposti in "*sotto -griglie*".// Questi sub-neuroni non vengono processati dall'algoritmo SOM, ma vengono utilizzati solo per campionare l'altezza del terreno sotto di loro.
Una volta terminato l'infittimento e l'ulteriore campionamento, si procede all'assegnazione dei tipi anche per i sub-neuroni.
I valori utilizzati per discriminare il tipo di un neurone variano in base alla dimensione n delle sotto-griglie:

$$plain\ slope\ sub\text{-}neuron = \frac{plainslope}{n}$$
e
$$hill\ slope\ sub\text{-}neuron = \frac{hillslope}{n}$$

Al termine di ciò per ogni sub-neurone non di tipo PLAIN si verifica che l'asse Z passante per la sua posizione interseca un qualsiasi edificio e se sì, quest'ultimo viene rimosso.

3.5 Costruzione del supporto visivo alla Griglia e uso dei colori

Ho sviluppato, come supporto al tool, un'interfaccia che permette la visualizzazione della griglia al termine dell'esecuzione, in modo tale da contestualizzare e verificare la correttezza e del risultato ottenuto.
Senza il supporto visivo infatti non si è in grado di capire né se il tool riesca correttamente ad analizzare il terreno, né se il terreno ha bisogno di modifiche per poter essere inserito in un ipotetico gioco.

Per fare ciò ho associato una sfera ad ogni neurone della griglia e li ho collegati, solo verticalmente e orizzontalmente, tramite dei ciclindri, andando così a creare una griglia che segue, in linea di massima, l'andamento delle varie pendenze del terreno.
L'uso dei colori invece è molto utile per il riconoscimento immediato del tipo di area a cui corrisponde una qualsiasi porzione di terreno, facilitando notevolemte la leggibilità del risultato del tool.

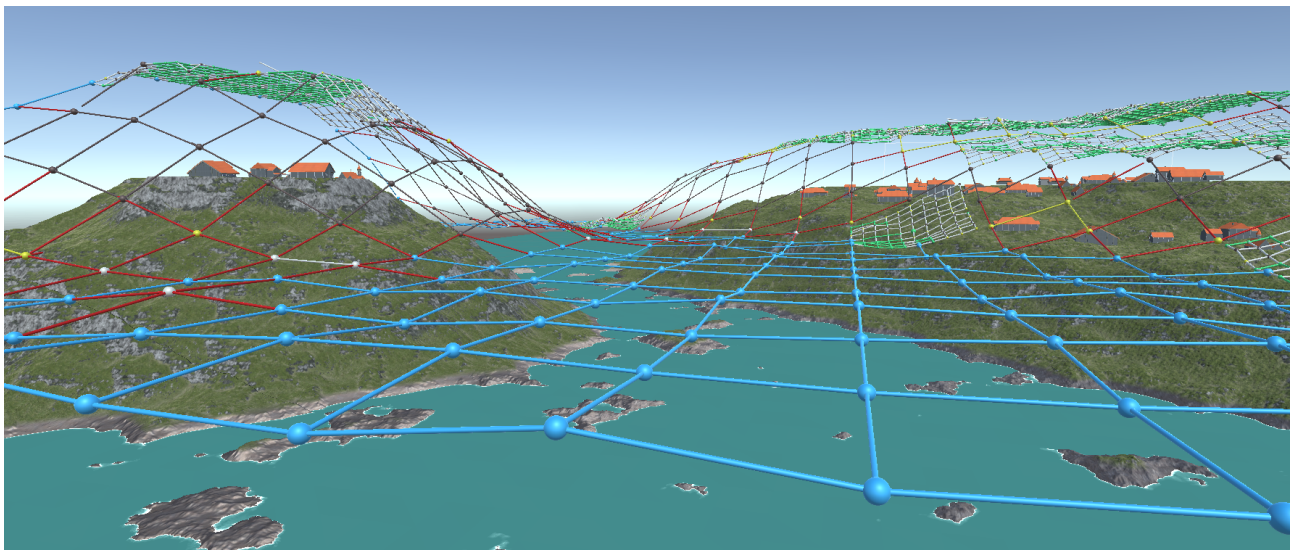

Figura 7: supporto visivo al tool

I colori utilizzati sono:

- AZZURRO per il tipo SEA
- BIANCO per il tipo SHORE
- VERDE per il tipo PLAIN
- GIALLO per il tipo HILL
- GRIGIO SCURO per il tipo UNSUITABLE

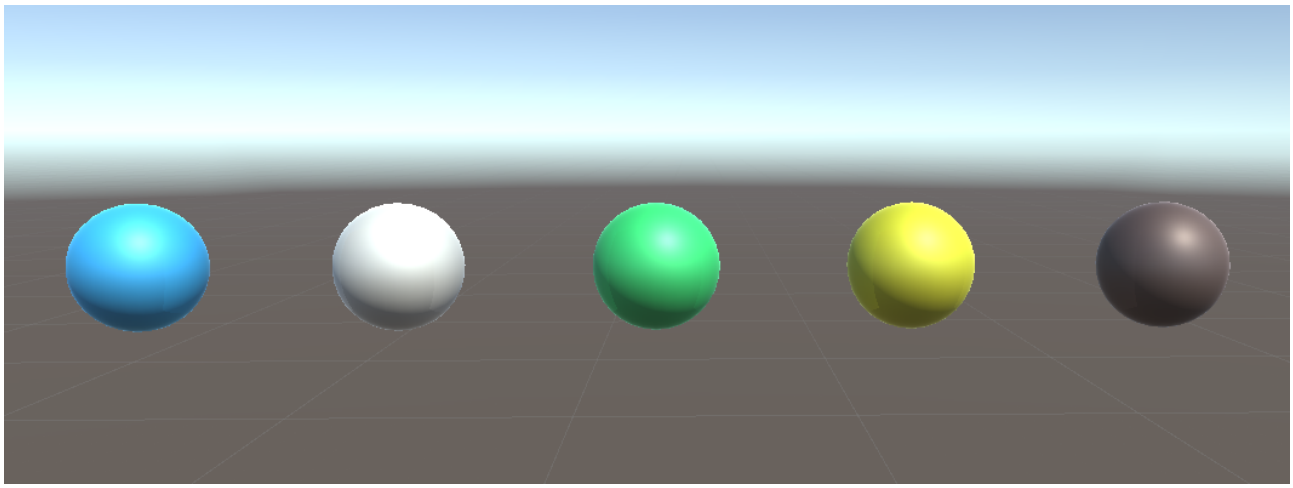

Figura 8: differenti tipi di neuroni

Nell'attribuire il colore corretto anche alle connessioni tra neuroni ho dovuto introdurre una regola per la quale un link tra 2 neuroni dello stesso tipo è dello stesso colore dei neuroni, mentre una connessione tra neuroni di tipo diverso è ROSSA (vedi Figura 7).

3.6 MapMagic e generazione procedurale del terreno

Nonostante il tool sia sviluppato per essere indipendente dal sistema di generazione procedurale del terreno, ho dovuto costruire un'interfaccia che comunicasse con il pacchetto MapMagic2 e ho dovuto impostare il grafo di generazione del terreno (strumento fornito da MapMagic) per dei terreni semi-realistici.

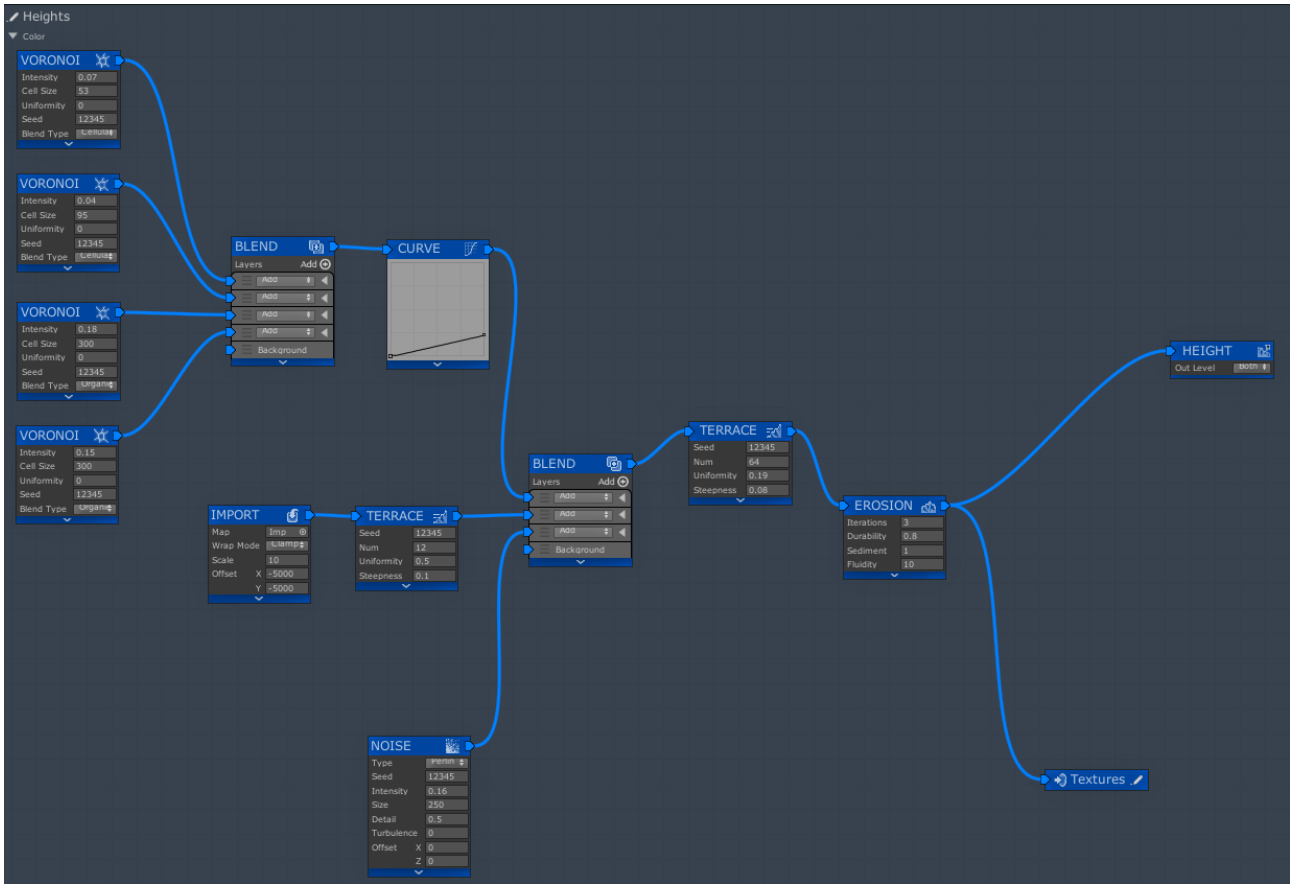

Figura 9: grafo della "forma" del terreno

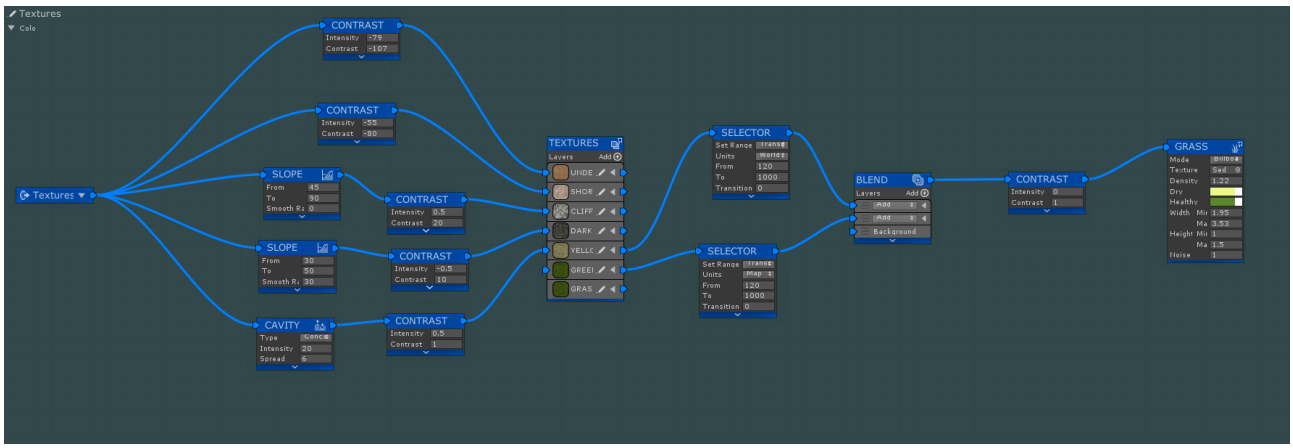

Figura 10: grafo delle textures del terreno