|  |
| --- |
| Projet XYZ |

Table des matières

[Analyse préliminaire 5](#_Toc511141175)

[Introduction 5](#_Toc511141176)

[Objectifs 5](#_Toc511141177)

[Planification initiale 7](#_Toc511141178)

[Analyse / Conception 7](#_Toc511141179)

[Concept 7](#_Toc511141180)

[Stratégie de test 7](#_Toc511141181)

[Risques techniques 7](#_Toc511141182)

[Planification 7](#_Toc511141183)

[Dossier de conception 7](#_Toc511141184)

[L’environement de travail. 8](#_Toc511141185)

[Maquettes 9](#_Toc511141186)

[MLD Utilisateur 10](#_Toc511141187)

[MLD Elément de la biosphère 11](#_Toc511141188)

[Structure du fichier de projet 11](#_Toc511141189)

[Techniques 12](#_Toc511141190)

[Placement de Objets 13](#_Toc511141191)

[Réalisation 13](#_Toc511141192)

[Dossier de réalisation 13](#_Toc511141193)

[Structure de fichier (Scripts) 13](#_Toc511141194)

[Librairies externes 15](#_Toc511141195)

[Description des tests effectués 16](#_Toc511141196)

[Erreurs restantes 16](#_Toc511141197)

[Liste des documents fournis 16](#_Toc511141198)

[Conclusions 16](#_Toc511141199)

[Annexes 17](#_Toc511141200)

[Résumé du rapport du TPI / version succincte de la documentation 17](#_Toc511141201)

[Sources – Bibliographie 17](#_Toc511141202)

[Journal de travail 17](#_Toc511141203)

[Manuel d'Installation 17](#_Toc511141204)

[Manuel d'Utilisation 17](#_Toc511141205)

[Archives du projet 17](#_Toc511141206)

*NOTE L’INTENTION DES UTILISATEURS DE CE CANEVAS:  
Toutes les parties en italiques sont là pour aider à comprendre ce qu’il faut mettre dans cette partie du document. Elles n’ont donc aucune raison d’être dans le document final.*

*De plus, en fonction du type de projet, il est tout à fait possible que certains chapitres ou paragraphes n’aient aucun sens. Dans ce cas il est recommandé de les retirer du document pour éviter de l’alourdir inutilement.*

# Analyse préliminaire

## *Introduction*

Dans le but de préparer notre TPI je réalise un projet de programmation portant sur la génération procédurale. Pour réaliser ce projet pour approfondir mes connaissances dans le moteur 3D qu’est Unity.

Unity met à disposition de nombreux élément pour réaliser mon projet, étant donné que j’ai déjà des connaissances intermédiaires du logiciel et de son API, le choix se justifie largement.

Le but de l’application sera de pouvoir générer un environnement selon les paramètres choisis par son utilisateur.

## *Objectifs*

L’utilisateur devra être capable :

* De générer un terrain en fonction des paramètres qu’il a choisi
* D’explorer l’environnement généré grâce à son clavier et sa souris et interagir avec les élément présent (arbres, plantes, rochers).
* D’enregistrer son projet pour le rouvrir plus tard
* De se connecter à un compte qu’il peut créer dans l’application

## *Planification initiale*

# Analyse / Conception

## *Concept*

## *Stratégie de test*

## Risques techniques

## *Planification*

## *Dossier de conception*

### L’environement de travail.

#### Unity

Unity est une application permettant de développer des jeux en 2D et en 3D pour toute la plateforme a dispositions. Enormément d’outils sont mis à disposition pour rendre le développement de jeux plus facile, et il est naturellement possible de développer soi-même des éléments qui ne sont pas mis à disposition. Le but de mon projet n’est pas de développer chaque algorithme ou développer chaque fonctionnalité, mais de les comprendre et d’apprendre à utiliser Unity d’une façon plus avancée.

Il est possible de créer des jeux sans même programmer ou en utilisant des blueprints pour implémenter une certaine logique, Si Unity ce limiterai à ça il n’y a aucun intérêt à choisir ce programme pour réaliser mon projet. Heureusement il est possible de développer des scripts qui seront compiler par Unity, ces derniers permettront de contrôler le comportement de certains objets dans la scène, ou encore développer des extensions pour l’éditeur pour rendre le flux de travail plus facile.

Pourquoi j’ai choisi Unity, Etant grandement intéresse par le développement de jeux-vidéo, j’essaie durant mon temps libre de maitriser ce logiciel dans le but développer des mini jeux, en plus de cela les langages de programmations utilisent pour rédiger les scripts sont en C# ou en javascript, deux langages que j’ai appris durant ma formation au CPNV.

#### Visual Studio Community

Un environnement de développement très répute permettant de développer des applications pour l’environnement .NET ainsi que de programmer dans divers langages comme le c++, … Il est possible d’utiliser quasiment n’importe quel éditeur de texte, IDE avec Unity mais la plupart des extensions pour déboguer ou faciliter le développement sont mis à disposition avec Visual Studio.

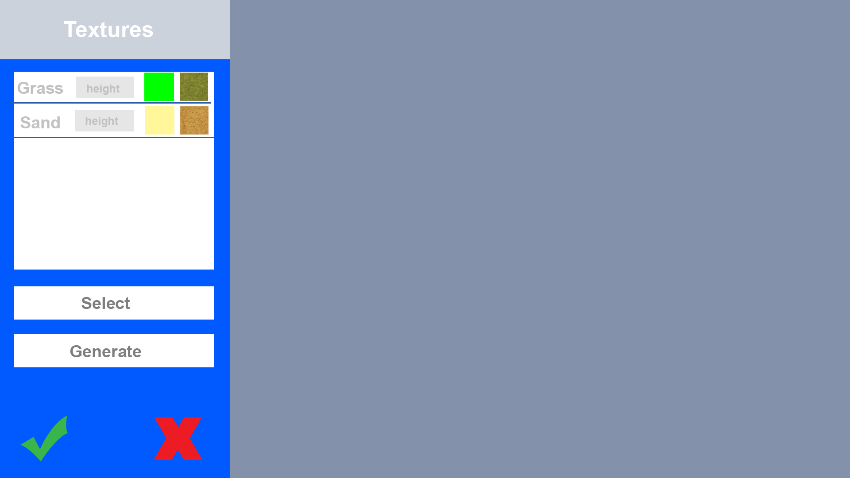
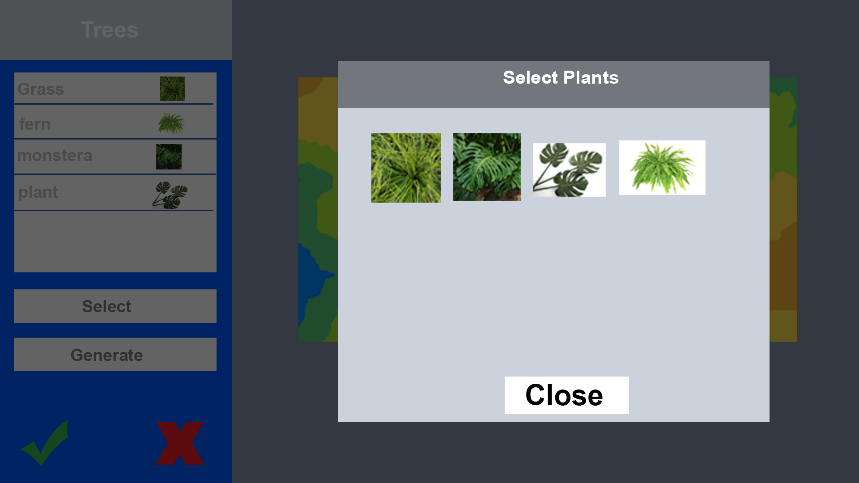
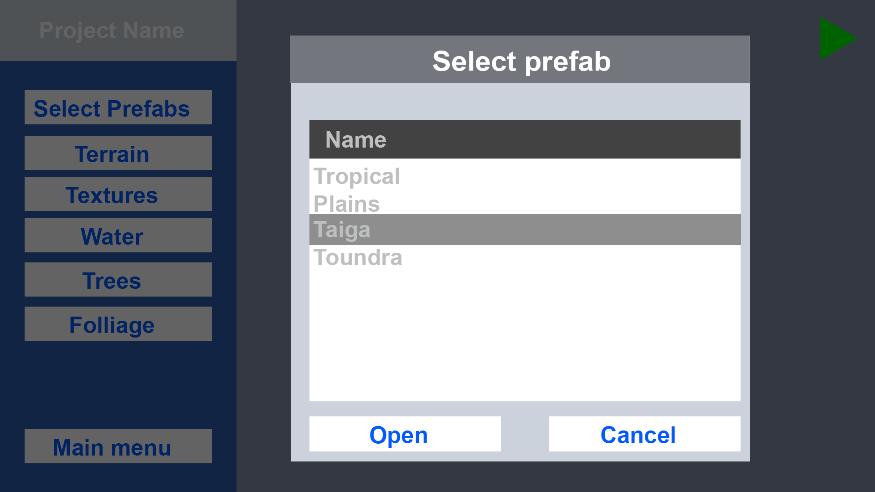
#### inVision

Une plateforme web qui permet de réaliser des maquettes pour des sites web, mon application ne sera pas une application web mais la réalisation de maquettes interactive est une chose bien utile qui me permet de démontrer mes intentions plus facilement.

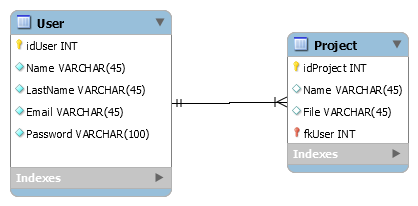
### Maquettes

Parmi ces étapes l’utilisateur pourra choisir des paramètres qui permettrons de générer le terrain désiré.





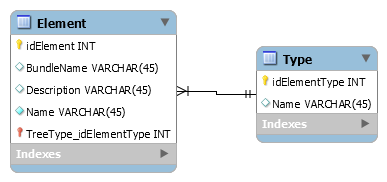
### MLD Utilisateur

L’utilisateur pourra aussi enregistrer ses projets dans une base de données

### MLD Elément de la biosphère

En ce qui concerne les modèles utilisés ils seront stockés aussi sur un Site Hébergeur, ou l’application auras l’autorisation de télécharger les « AssetBundles [[1]](#footnote-1)» nécessaire.

Pour les éléments avec lequel l’utilisateur pourra interagir une autre base de donnée est créé pour stocker les détails des biomes. La composition de cette base de données dépend grandement de la granularité définie, car il est possible d’aller très loin dans la hiérarchie de la biosphère.



### Fichier de projet (Terrain généré)

#NoiseMap

-Width=1024

-Height=1024

-Lacunarity=1.5

-Persistence=0.5

-Seed=1024

-WorldScale=100

#SplatMap

-Total=2

<Texture>

-Name=TerrainGrass\_basecolor

-mode=HeightRange

-minheight=30

-maxheight=100

</Texture>

<Texture>

-Name=TerrainDirt\_basecolor

-mode=HeightRange

-minheight=0

-maxheight=31

</Texture>

#Lakes

-OceanLevel=30

#Vegetation

-Total=11

<Plant>

-Name=FAE\_Birch\_C

-mode=HeightRange

-minheight=30

-maxheight=70

-density=50

-clump=40

-resolution=100

</Plant>

Pour pouvoir enregistrer le terrain il faut trouver un moyen pour enregistrer les données pour générer exactement le même terrain.

Pour réduire la complexité de la base de données le projet sera exporté dans un fichier qui contient tous les détails de la scène. La syntaxe du fichier pourra encore évoluer durant le projet mais le principe est de pouvoir enregistrer chaque paramètre rentré par l’utilisateur, les lister par catégories

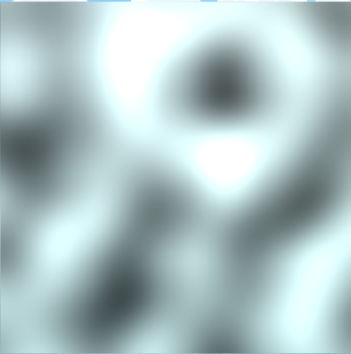
### Techniques

#### Générer un terrain

Un terrain génère procéduralement n’est pas dessiner par l’utilisateur même. Plusieurs règles sont définies qui permettent au programme de savoir ce qu’il doit faire un élément impératif à avoir est un HeightMap ou encore NoiseMap. C’est une image sur laquelle un bruit est génère « aléatoirement ». Effectivement ce n’est pas totalement aléatoire sinon on obtiendra une Image pareille.

Le but de ce HeightMap est d’avoir une représentation d’un terrain vu d’en haut en 2D ou les pixels noirs sont les points les plus profond et les pixels blanc les points les plus haut.

Figure 1 Bruit aleatoire



Pour rendre cette génération le moins aléatoire possible il est possible d’utiliser un algorithme développé par ken Perlin, le PerlinNoise. Ce type de bruit permet de générer des valeurs qui ont une cohérence entre eux et qui permet d’obtenir un HeightMap tel quel.

Figure 2 Perlin Noise

De cet image il est possible de générer un terrain en 3D avec un composant mis à disposition par Unity. En fournissant les données obtenues du HeightMap l’outil terrain de Unity peut generer un mesh en 3D.

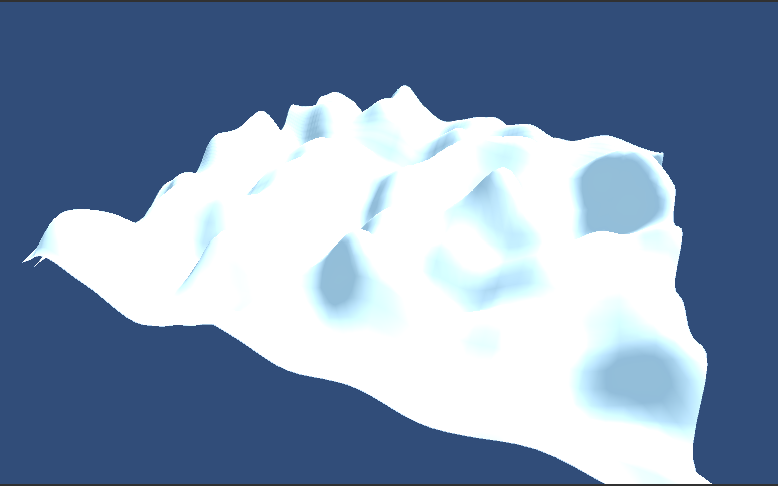
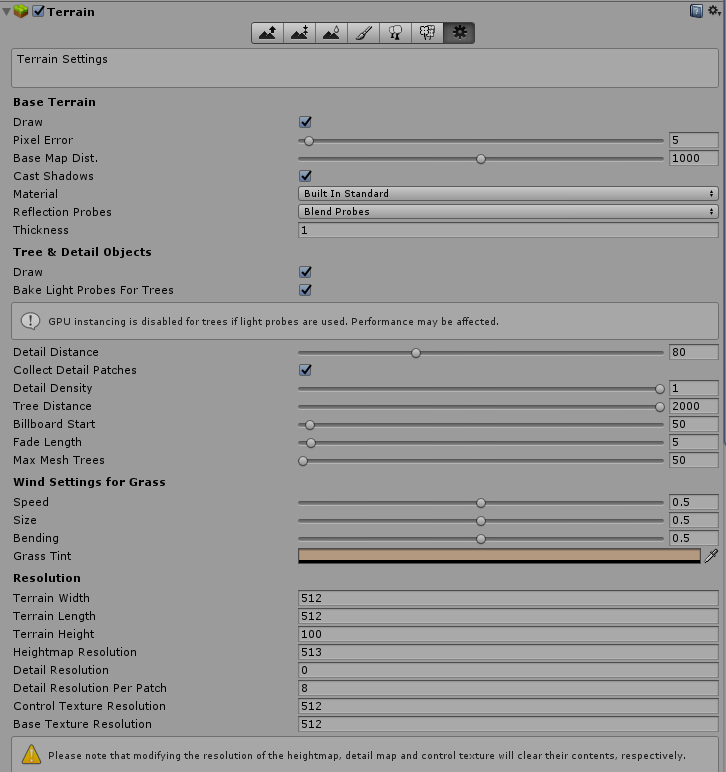


Figure 3 Terrain générer à partir d'un heightMap

Figure 4 Outil terrain de unity

#### Générer les textures

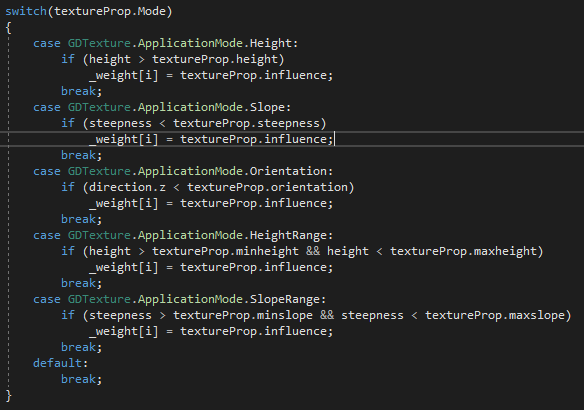
Les textures sont les éléments qui sont directement collées sur le terrain de base (Fig. 3) Avec l’outil terrain il est possible de peindre les textures désirées sur le terrain, mais étant donne que mon application n’est pas une extension de l’éditeur Unity mais bien une application indépendante il faut écrire un script qui permet de placer des textures en exploitant l’outil terrain de Unity, en définissant diverses règles pour que les textures ne se placent pas n’importe comment. Les règles de base que je désire utiliser sont la hauteur, la pente et une gamme d’hauteur (min, max).

Figure 5 regles de generation de textures

* Créer un tableau avec comme contenu des textures utilisables
* Récupérer les informations sur la pente, hauteur, orientation du monde 3D
* Appliquer les textures sur le terrain 3D selon le mode d’application

[Citez votre source ici.]

### Placement de Objets

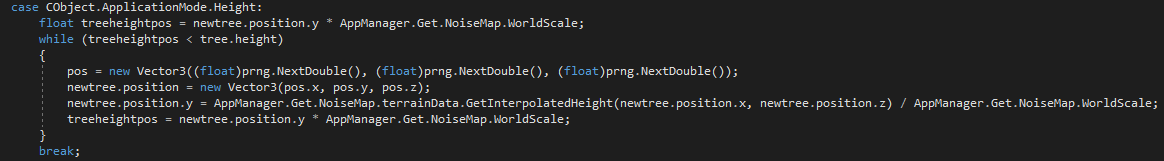
Comme les textures L’outil terrain met à disposition une fonctionnalité pour peindre des objets sur le terrain et de même il est possible de réaliser cela par script. Le principe est de générer des positions aléatoires en respectant les règles définies comme pour les textures. Je suis parti du principe que tant une position générer ne correspond pas à la règle il faut la régénérer.

Figure 6 Regle de generation de objet

Le placement d’objet avec l’outil terrain de Unity est très rapide mais vient avec quelques inconvénients, il est compliqué de gérer les interactions avec les objets et il faut régénérer à chaque modification les collisions.

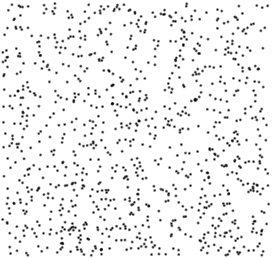
J’ai donc effectué des recherches sur des moyens pour placer des objets sur un terrain aléatoirement, l’outil terrain de Unity a son propre algorithme de placement, si je générais des positions aléatoires il y aurait beaucoup d’incohérence dans le positionnement des objets, c’est là que j’ai trouvé un algorithme de placement appelé poisson disk sampling. Qui n’est pas complique à implémente et permet de générer des positions pseudo-aléatoires.

Figure 7 Poisson disk sampling

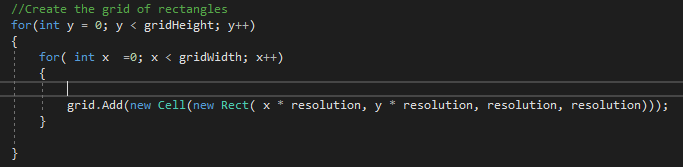
Le principe est de diviser ma carte en cellules dans lesquels je vais générer un point aléatoire. Ensuite autour de ce point je vais générer d’autre positions conforme à une distance minimale et si cette position se trouve dans la cellule d’origine.

Figure 8 Generer des cellules

# 

-Diviser le terrain 3d en un tableau de cellules

-Générer une position aléatoire dans chaque cellule

-Ajouter les positions générer précédemment dans un tableau dynamique

-Pour chaque position dans le tableau de points générer un nombre de points définis autour de ce point

-sauver le tableau de points générer pour être réutilisable plus tard

[Citez votre source ici.]

Figure generer un nombre de point autour d'un point aleatoire

Figure 10 Générer une position aléatoire dans une cellule

# 

Figure 11 Valider les positions generee

Une validation des positions est nécessaire sinon les objets se trouveront n’importe où sur le monde une deuxième liste de position est donc générée qui contiendra les positions valides ou il est possible de placer un objet spécifique.

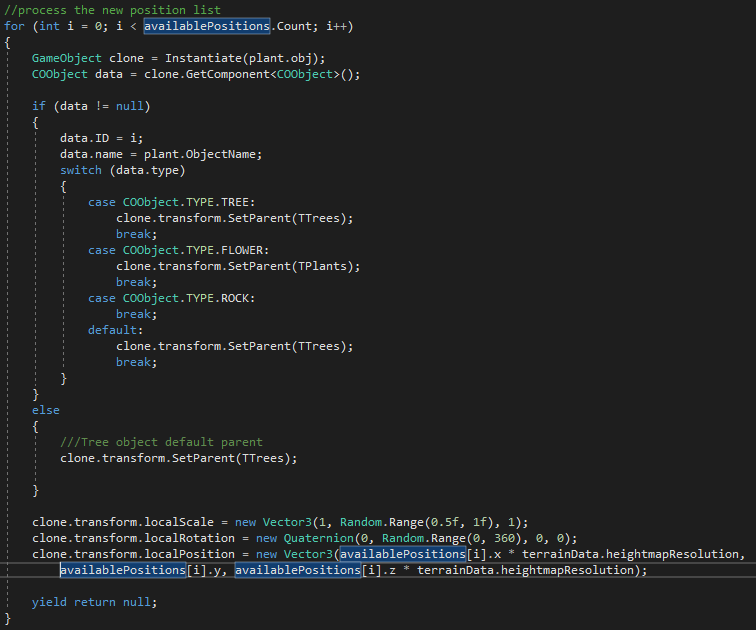
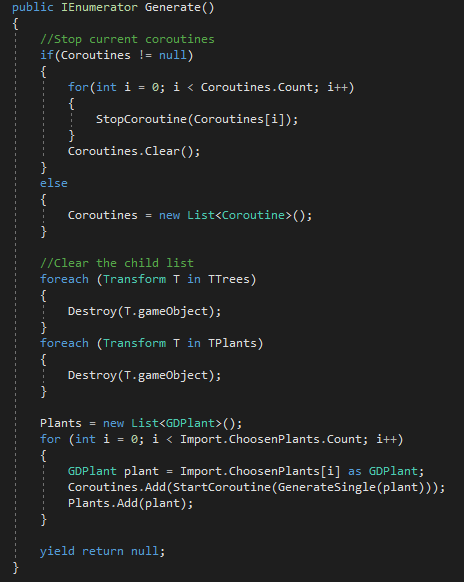


Figure 12 Creer les objets

Lors de mes précédentes tentatives de génération d’objet j’utilisais les méthodes standard pour générer un objet ce qui causait l’application de geler lors de la génération et cela durant un temps indéterminé car cela dépendait de la taille de la carte, du nombre d’élément qui sont générer.

Pour remédier a cela j’utilise les Coroutines cela permet créer un « Thread » à part qui tournera en même temps que la boucle principale du jeu et permettra d’exécuter du code simultanément.

La liste des coroutines représentes le nombre de plantes qui sont générée en même temps sinon il faudra attendre la fin de chaque génération pour en relancer une nouvelle.

Figure 13 Coroutines

Les placements de n’importe quel objet(naturel) sera procédée avec cet algorithme

# Réalisation

## *Dossier de réalisation*

### Structure de fichier (Scripts)

* Assets : Dossier contenant tout le script rédige par le développeur
  + Scripts
    - Algorithms
      * Cell.cs : Représente un rectangle
      * PoissonDisk.cs : Algorithme de placement
    - AssetLoader
      * BundleLoader.cs : Charge des assets depuis un fichier
      * ChooseAssets : Permet de gérer la sélection des Assets
    - Camera
      * FreeLookCamera.cs : Gérer le déplacement libre dans la scene
    - Custom Object : Des scripts qui sont attache à des prefabs, ces donnees sont exploitées pour interagir avec un Object spécifique dans la scène
      * COFlower.cs : Objet de type fleur
      * COObject.cs : Objet de base
      * COTree.cs : Objet de type arbre
    - Events
      * ListEvent.cs : Gere les événements sur une liste
      * PlayerController.cs : Gere l’événement avec un personnage
      * UserInput.cs : Gérer les évènement Clavier/Souris
    - Génération Data : Données utilisées pour générer des objets
      * GDBundle.cs : Données d’un groupe d’objet à charger
      * GDObject.cs : Données d’un objet de base
      * GDPlant.cs : Données pour la génération de plantes
      * GDRock.cs : Données pour la génération de rocher
      * GDTexture.cs : Données pour la génération de texture
    - Generator
      * LakeGenerator.cs : Génère des lacs
      * NoiseGenerator.cs : Génère le bruit/HeightMap
      * TextureGenerator.cs : Génère les textures
      * VegetationGenerator.cs : Génère la végétation
    - Managers
      * AppManager.cs : Contient des variables globales et permet de faciliter l’interaction entre les différentes classes
      * UIManager.cs : permet de gerer l’interaction avec l’interface graphique
    - UI
      * List
        + ListElement.cs : Un élément qui fait partie d’une liste
        + ListManager.cs : Permet de gérer plusieurs ListElement
      * Panel
        + NoiseUI.cs : Données de l’interface pour gérer la generation de Bruit
        + TreeUI.cs : Donnes de l’interface pour gérer la generation de végétation
        + WaterUI.cs : Données de l’interface pour gérer la generation des lacs
      * TabItem
        + SwitchTab.cs
        + TabItem.cs
      * EditorNavigation.cs
      * OpenAssetSelection.cs
      * PlantUI.cs
      * TextureUI.cs

### Librairies externes

#### Fantasy Adventure Environment

Je n’ai pas exactement utilise des librairies externes, mais j’ai utilisé un pack que j’ai acheté sur l’asset Store de Unity qui met à disposition tous les élément pour générer un terrain, ce pack contient comme élément

* 17 types de plantes
* 12 types d’arbres
* 6 matériaux pour arbres
* 6 matériaux pour terrain
* 7 rochers
* 4 amas de rocher
* Une skybox
* Un effet de lentille
* Effet de brouillard
* Effet de particule, feuille tombante, neige
* Un pack de post processing
* Un shader d’eau

Et les fonctionnalités(scripts) mis à disposition sont

* Animation de vent sur la végétation
* Déplacement de la végétation lors du déplacement d’un joueur
* Fusion de couleurs entre les plantes et la texture du sol
* Un système de LOD (Levels of Details)
* Des billards pour les arbres pour plus de performances

#### Unity-Chan

Pour le mode simulation et la gestion de déplacement d’un personnage j’utilise un modèle déjà prefait pour Unity.

Unity-chan est la mascotte de Unity et est gratuit sur l’asset store de nombreuse animation sont mis a disposition même si les seuls nécessaire pour mon projet sont les élément de déplacement

## *Description des tests effectués*

## *Erreurs restantes*

## *Liste des documents fournis*

# Conclusions

# Annexes

## *Résumé du rapport du TPI / version succincte de la documentation*

## *Sources – Bibliographie*

## *Journal de travail*

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Durée** | **Activité** | **Remarques** |
|  |  |  |  |
|  |  |  |  |

## 

## *Manuel d'Installation*

## *Manuel d'Utilisation*

## *Archives du projet*

1. AssetBundle : Terme utilisé par Unity pour définir un fichier qui contient tous les assets disponibles pour l’application [↑](#footnote-ref-1)