420-C61-IN

PROJET SYNTHÈSE

Présenté par :

Amine Fanid

Henrick Baril

Rapport Final

Table des matières

[Présentation générale 3](#_Toc167403665)

[Résumé du développement 3](#_Toc167403666)

[1. 3](#_Toc167403667)

[2. 3](#_Toc167403668)

[Fonctionnalités 4](#_Toc167403669)

[Parfaitement Fonctionnelle 4](#_Toc167403670)

[Partiellement Fonctionnelle 4](#_Toc167403671)

[Fonctionnelle avec limitations 4](#_Toc167403672)

[Identification des éléments techniques 4](#_Toc167403673)

[Interface graphique utilisateur 4](#_Toc167403674)

[Données persistantes 4](#_Toc167403675)

[Structures de données 5](#_Toc167403676)

[Patrons de conception 5](#_Toc167403677)

[Bibliothèque, algorithme, mathématique et regex 6](#_Toc167403678)

[Améliorations possibles 6](#_Toc167403679)

[Auto-évaluation individuelle 7](#_Toc167403680)

[Henrick 7](#_Toc167403681)

[Amine 7](#_Toc167403682)

[Auto-évaluation d’équipe 8](#_Toc167403683)

[Amine : 8](#_Toc167403684)

[Henrick : 8](#_Toc167403685)

[Références : 9](#_Toc167403686)

# Présentation générale

« Geek’s Legacy » est un jeu de plate-forme 2D dont le but est de récolter des ressources dans une carte généré aléatoirement afin d’aller vaincre les différent ennemies et « boss » du jeu afin que le personnage principal, le joueur, retrouve l’item mythique qui lui été volé, sa carte graphique.

Dans ce jeu, il est possible de récolter des items ainsi que de les lâcher au sol. De plus, il est aussi possible de miner les différents matériaux du monde.

Les interactions principales sont basées sur le clavier et la souris. Le clavier sert à mouvoir le joueur dans le monde. Pour la souris, celle-ci sert entre autres à utiliser les armes/outils.

Il est aussi possible pour les joueurs d’enregistrer leur carte sur laquelle ils jouent.

# Résumé du développement

## 1.

En général, la majorité de nos objectifs « essentiel » ont été implémenté. Cependant, nous avons eu quelques problèmes avec quelques détails durant le développement. En effet, la librairie contenant la génération procédurale de la carte fut un grand défi de généralisation de notre algorithme. Nous avons ainsi dû, par cause de manque de temps, délaisser certaines de nos fonctionnalités. Par exemple, nous n’avons pas implémenté le procédé de fabrication d’objet et donc, par le fait même, nous n’avons pas utiliser la structure de données « Graphe ». Aussi, nous avons pu implémenter qu’un seul ennemi et aucun boss.

## 2.

Nous avons eu des soucis avec la sérialisation de nos données dans notre fichier JSON, qui agit à titre de base de données. Ainsi, nous avons eu à faire beaucoup de refactorisation, et ce même très tard dans la session.

Le développement de la machine à état fut un événement marquant de la production de notre jeu. En effet, celle-ci a nécessité 30 heures de plus que prévu initialement. Ce qui nous a ralenti pour toutes les autres fonctionnalités.

# Fonctionnalités

## Parfaitement Fonctionnelle

* Mouvement du joueur (gauche, droite, sauter, arrêt).
* Génération de la carte de façon aléatoire (biomes, cavernes, montagnes).
* Enregistrement de la carte selon le compte du joueur.
* Ordre d’affichage des scènes du jeu (login/création de compte, jeu)

## Partiellement Fonctionnelle

* Enregistrement de l’inventaire du joueur.
* Interface utilisateur (inventaire, barre de vie, bouton).
* Ennemi et leurs mouvements (déplacement et détection du joueur).

## Fonctionnelle avec limitations

* Utilisation de l’inventaire
* Détection des collisions (attaque, perte de vie, etc.).

# Identification des éléments techniques

## Interface graphique utilisateur

* Interface de Login : Initialiser et déclarer à l’intérieur de la scène « LoginScene » dans Unity.
* Interface d’inventaire : Initialiser et déclarer à l’intérieur de la scène « MapGenTemp » dans Unity.
* Interface de vie : Initialiser et déclarer à l’intérieur de la scène « MapGenTemp » dans Unity.

## Données persistantes

* Stockage de certaines données dans un fichier JSON.
* Dossier de scripts C# nommé GameSaveScripts. Contient les scripts qui gèrent la base de données.
* Dans le fichier WriteOrReadUser.cs, aux lignes 71 et 72. On transforme une liste de données d’usager en format JSON et on écrit le texte JSON dans un fichier.
* Dans le fichier LoadUserData.cs, aux lignes 19 et 20, on lit tout le contenu du fichier JSON, et on retourne le tout dans une string.

## Structures de données

* Dictionnaire :
  + L’inventaire du joueur par exemple, utilise un dictionnaire pour stocker le nombre de fois qu’on a un certain élément dans notre inventaire. Ex : un élément de « dirt », 15 fois.
  + Ligne 10 du fichier CharacterInventory.cs, ainsi que dans plusieurs autres lignes du même fichier où on interagit avec ce dictionnaire.
* Matrice :
  + Carte du monde stocké dans une matrice 2D.
  + Dans le fichier ProceduralGeneration.cs, à la ligne 91, et dans plusieurs autres lignes du même fichier, car c’est notre libraire qui gère la carte du monde généré aléatoirement.
* Liste :
  + Utilisé à plusieurs reprises, ex : Liste qui garde en mémoire les coordonnées de la surface de la carte, afin de pouvoir générer le personnage au-dessus de la carte.
* Liste chainée :
  + Utilisé pour stocker les ennemies ainsi que leurs données (Vie, butins, etc.).
  + Déclaration : lignes 18 à 30 dans le script « EnnemiesSpawn ».
  + Utilisation : lignes 81 à 140 dans le script « EnnemiesSpawn » dans la fonction « AddNode() ».

## Patrons de conception

* Observer :
  + Utilisé dans le script HealthBar.cs et Player.cs. Adapte le visuel de la barre de vie du joueur selon un écouter placer sur la vie du joueur dans l’objet player.
  + Instanciation : lignes 47 à 57 dans le script « EventManager ».
  + Usage : ligne 24 dans le script « HealthBar ».
* Fabrique :
  + Utilisé dans le script ItemFactory.cs, nous permet de créer un item selon son nom (en string) d’à peu près n’importe où dans le code.
* État :
  + Utiliser dans les scripts « State », « FSM » et « Slime » permettant de gérer des changements d’états en fonction de certaines conditions. Permet une meilleure gestion du comportement de nos ennemies.

## Bibliothèque, algorithme, mathématique et regex

* Bibliothèque :
  + ProceduralGeneration.cs peut être utilsé pour reproduire une carte pour un jeu de plateforme 2D du style de Terraria ou Geek’s Legacy.

(Déclaration lignes 122 à 133 et usage ligne 28 dans le script GeeksLegacyLauncher).

* Algorithme :
  + On utilise du PerlinNoise, du Cellular Automata, ainsi que la classe Random dans ProceduralGeneration.cs.

(Usage ligne 228 dans le script ProceduralGeneration).

* Mathématique :
  + Utiliser pour définir les mouvements (physique).

(Usage lignes 87 à 90 dans le script « TileDetection »).

* Regex :
  + Fonction checkPassword dans le script WriteOrReadUser.cs.

(Usage ligne 24 dans le script WriteOrReadUser).

# Améliorations possibles

* La sérialisation dans le fichier JSON des éléments de notre jeu. Tel que l’inventaire, par exemple, qu’on ne peut pas sérialiser pour le moment, puisqu’il contient des types composés tel que des dictionnaires. Nous avons pu adapter la carte du jeu pour la sérialiser, mais on a manqué de temps pour l’inventaire.
* On pourrait mettre plus de temps dans des patrons de conception tel que le Singleton, qui aurait été idéal pour notre code qui accède au fichier JSON. Au lieu d’accéder à plusieurs différents scripts dans plusieurs parties de notre code.
* La communication avec l’usager, par exemple : des messages d’erreurs pour l’usager, lorsque l’usager ne joue pas correctement. Comme des messages d’erreurs dans le terminal, mais pour l’usager.

# Auto-évaluation individuelle

## Henrick

Ce projet reflète notre capacité d’adaptation au nouveau langage et au projet de plus grande envergure. Selon moi, les cours de gestion de projet et le fais que nous apprenions plusieurs langages dans la technique nous a permis de réussir ce projet.

Ce projet me démontre à quel point, en si peu de temps, du fait d’être à l’intensif, nous pouvons effectuer des projets de cette envergure. En effet, je n’aurais jamais pensé être capable de fournir ce genre de jeu si on me l’avait demandé à la première session.

Deux éléments que je retiens le plus de ce projet sont :

* + - Aspect humain :
      * Notre capacité de communication et d’enthousiasme pour le projet.
    - Aspect technique :
      * Prévoir plus de temps pour la planification, car celle-ci est primordial.

## Amine

Mon opinion est assez partagée concernant ce projet de synthèse. J’ai pu utiliser plusieurs notions que j’ai appris dans la technique. Cependant, travailler dans un environnement de jeu vidéo ou on utilise Unity et C# ne m’était pas du tout familier, et j’ai eu quelques de difficultés à m’y adapter.

Malgré tout, ce que j’ai acquis dans la technique, m’a vraiment servi pour ce projet. Ma capacité d’adaptation qui fut affuté pendant la technique, et les connaissances que j’ai acquis en programmation orienté objet, en organisation, en patron de conception, etc. ont bien été utilisé dans ce projet. Je suis amplement satisfait, et fier de ce que j’ai appris pendant la technique et de ce que je suis capable d’accomplir maintenant.

Deux éléments que je retiens le plus de ce projet sont :

* + - Aspect humain :
      * Notre enthousiasme face au défi qu’est notre projet de synthèse.
    - Aspect technique :
      * Ne jamais lésiner sur la planification d’un projet.

# Auto-évaluation d’équipe

## Amine :

On aurait vraiment profité d’une meilleure communication, par exemple, faire plus de SCRUM, ou des SCRUM plus complets. Il nous est arrivé de simplement coder chacun de notre côté, sans communiquer pendant environ 2 semaines, et d’être perdu dans le code de l’autre, car on ne communiquait pas ce qu’on faisait. On a aussi eu quelques MERGE CONFLICTS sur GITHUB. On aurait profité, aussi, à travailler plus souvent l’un à côté de l’autre, non seulement pour la communication, mais aussi pour l’entraide.

D’un côté plutôt positif, notre équipe fut quand même très soudé pour ce projet, on s’est vraiment entraidé. D’ailleurs, les lacunes de l’un, étaient supportés par l’autres, et on prenait notre temps pour régler ensemble nos soucis de codes, tel que des bugs où problèmes de conception.

## Henrick :

Nos points fort ont été notre dévouement, notre capacité d’adaptation et notre flexibilité. En effet, chacun a su mettre tous les efforts possibles dans le projet tout en s’adaptant au travers des défis que peux représenter un nouveau langage et un nouvel environnement de travail qu’est Unity.

Cependant, notre manque de communication dans certain moment clé nous a causé des embuches. En effet, nous aurions pu profiter de plus de rapidité de production de code en ayant une communication plus fréquente et structuré.

En bref, notre travail d’équipe c’est bien dérouler, mais pourrait être améliorer pour mieux se compléter.

# Références :

* [https://youtu.be/neTvQEDhZZM?si=7Bdp6xrINJxV7MQU](https://youtu.be/neTvQEDhZZM?si=7Bdp6xrINJxV7MQU%20)

(Vidéo de ChronoABI, pour le script ProceduralGeneration.cs)

* [https://youtu.be/l5KVBDOsHfg?si=\_RxjTLFp0sT07Zml](https://youtu.be/l5KVBDOsHfg?si=_RxjTLFp0sT07Zml%20)

(Vidéo de ChronoABI, pour le script ProceduralGeneration.cs)

* <https://docs.unity3d.com/Manual/index.html>

(Docs Unity)

* <https://www.youtube.com/watch?v=g0k4XStzkLQ&t=2s>

(Vidéo de Unity3D School, pour le fichier JSON)

* <https://youtu.be/MyVY-y_jK1I?si=MCug8hPOaxgrvybi>

(Vidéo de Ketra Games, pour utiliser Vector3.Lerp dans le script MaterialsBehaviour.cs)

* <https://youtu.be/zABMvbTyEK8?si=Q8fnf8rccELpp3vk>

(Vidéo de Pxl Dev, a été utile pour le script clickDetection.cs)

* <https://faramira.com/finite-state-machine-using-csharp-delegates-in-unity/>

(Forum de Farima, a été utile pour le squelette de la FSM)

* <https://bitbucket.org/akilea/foretmagiqueint/src/master/Assets/Script/ForceAnimationVirtualJoystick.cs>

(BitBucket de Marc-André Lavoie, a été utile pour le script « ForceAnimationVirtualJoystick »)