**RAPPORT DE STAGE**

“Clustering et visualisation des trajectoires 3D“

Mihai ANCA

21905844

Licence Informatique

**SOMMAIRE**

1. Introduction………………………………………………………………….3
2. L’algorithme k-means clustering……………………………...……………..4
3. L’algorithme DTW…………………………………………………………..7
4. Le passage vers k-means++.............................................................................9
5. La visualisation en IATK et la démo Unity……………….………………..11
6. Conclusion………………………………………………………………….13
7. Bibliographie / Sources photo……………………..……………………….15
8. **Introduction**

L’une de mes plus grandes passions est représentée par les jeux vidéo. Vers la fin de 2016, j’ai découvert le fait qu’il y a un média plus interactif que les films, la musique ou la peinture que j’avais presque complètement ignoré avant. L’interactivité vient d’un mélange de plusieurs arts qu’on nous offre pour avoir une expérience unique selon nos envies, nos choix, nos compétences. Cet aspect m’a fasciné et étant donné que j’ai choisi l’informatique en licence, j’ai envisagé un parcours professionnel dans ce domaine.

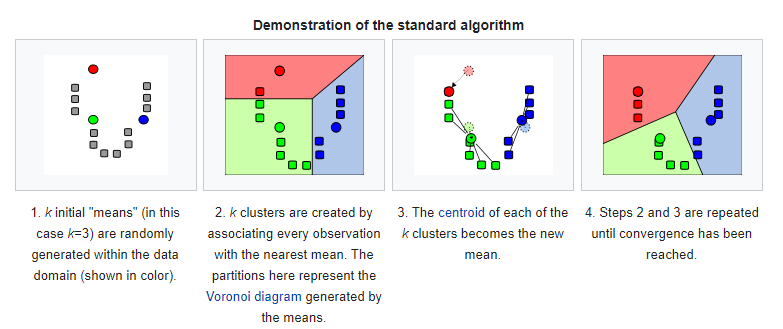
Après avoir appris l’existence des stages en licence, j’ai fait de nombreux essais pour trouver des stages de courte durée dans des entreprises de jeux vidéo de Bordeaux et ailleurs, mais malheureusement sans succès. Soit elles n'offrent pas de stages de courte durée pour des étudiants en licence, soit elles n’en offrent pas du tout à cause de la pandémie. A l’aide de mon enseignant référent et grâce à mes notes des 3 semestres précédents, j’ai réussi à trouver un stage pertinent chez INRIA Bordeaux. Il s’agit d’un laboratoire de recherche en informatique composé par 20 équipes-projets et plus de 300 personnes. Les axes principaux de recherche sont les Mégadonnées, l’Apprentissage automatique et la santé numérique.[[1]](#xpoawmmdihto)

J’ai été mis en contact avec mon futur tuteur de stage, M. Arnaud PROUZEAU, qui m’a accueilli dans l’équipe POTIOC. Elle est la composante qui s’occupe des méthodes non-conventionnelles d’interaction avec l’ordinateur (comme les interfaces AR et VR). Après quelques discussions, nous avons choisi un sujet qui me bénéficierait sans s’éloigner des sujets qui intéressent l’équipe: la représentation visuelle des trajectoires clusterisées à l’aide d’Unity et de l’outil IATK.

J’ai accepté ce sujet comme un moyen de commencer à apprendre Unity pour développer des jeux vidéo plus tard. Avec ce but en tête, avant de commencer le stage lui-même, mon tuteur m’a recommandé quelques tutoriels[[2]](#vpzcx9o6rryo)[[3]](#o7i9wlj2hvu)[[4]](#rj7ut75kll9a) pour m’habituer à utiliser l’interface Unity et à programmer en paradigme orientée objet. Pour ceci, j’ai suivi le tutoriel de base “Roll-a-ball” proposé par Unity, un tutoriel Java pour apprendre la POO et un dernier pour un jeu “Tower Defense” pour aller plus loin.

1. **L’algorithme k-means clustering**

Pour la première partie du stage, je me suis concentré sur la partie clustering des trajectoires, plus théorique et moins visuelle. Ceci m’a donné le temps requis pour apprendre à coder plus aisément en C# avec les bibliothèques fournies par Unity. Ma première tâche a été de comprendre et de mettre en place l’algorithme “k-means clustering” pour organiser des points 2D en k zones distinctes.



1. L’Algorithme k-means expliqué[[5]](#ylyjnsxrvr81)

L’algorithme est assez facile à comprendre et à implémenter. On va considérer qu’on a n points et un entier positif k <= n en entrée, le nombre de clusters désirés.

1. On choisit aléatoirement k points. Ceux-ci représentent les centroïdes initiaux des k clusters.
2. Pour chacun des autres n-k points, on va trouver le centroïde vers lequel la distance est minimale et on va mettre ce point dans le cluster correspondant.
3. Pour chaque cluster, on recalcule le centroïde comme étant la moyenne des points du cluster courant.
4. On reprend les pas 2 et 3 jusqu’au moment où les clusters se stabilisent, i.e. entre 2 itérations successives, la configuration de chaque cluster ne change plus.

La partie plus complexe a été la mise en pratique de l’algorithme avec les outils que j’ai eu à disposition, avec plusieurs choix à faire pour trouver la meilleure implémentation.

Etant donné que je ne m'occupais pas encore de la visualisation de ce type de données, on a choisi de lire et écrire à l’aide du format de fichiers .csv (valeurs séparées par des virgules) et de visualiser le résultat final à l’aide d’un script en langage R fourni par mon tuteur de stage.

Pour organiser le code, j’ai choisi de créer une classe PCluster:



**public class PCluster**

{

**public** Vector2 Centroid;

**public** List<Vector2> Points;

}

Avec un Vector2 qui sert comme centroïde et une liste de Vector2 en tant que liste de points contenus dans le cluster, j’ai écrit mon code principal dans une classe Point. Je l’ai découpé en plusieurs fonctions pour une meilleure factorisation du code:



**public void kMeansPClustering**(**uint** k);

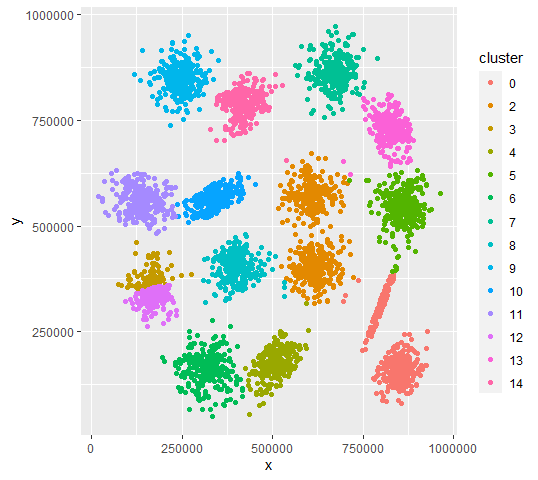
**public void WriteClusters**(List<PCluster> Clusters);

**public** Tuple<List<Vector2>, **float**, **float**, **float**, **float**> **ReadPoints**();

Un appel de la fonction implicite Start() déclenche l’algorithme k-means avec une valeur k en entrée. On lit tous les points contenus dans un fichier .csv avec le format “x,y” et on renvoie un tuple contenant la liste des points et les 4 valeurs flottantes représentant xmin, xmax, ymin et ymax.

Pour rester dans un rectangle restreint, on choisit les centroïdes comme des points aléatoires (xi entre xmin et xmax et yi entre ymin et ymax). On crée k instances de la classe PCluster et on initialise les centroïdes avec les points aléatoires. Avec cette initialisation, on répète les pas 2 et 3 du schéma (a) tant que les clusters restent instables et qu’on n’a pas atteint les 1000 itérations. En effet, on se rend compte qu’il y a parfois plusieurs états stables entre lesquels les clusters fluctuent, ce qui justifie le choix d’une limite arbitraire de 1000 itérations. A la fin, on écrit les clusters dans un fichier .csv avec le format “x,y,cluster”.

Pour pouvoir tester ce type d’algorithme sur une grande échelle, on a trouvé des fichiers avec des milliers de coordonnées[[6]](#lj5507yvqa3s), on les a transformés au format spécifique à l’entrée. Avec le script R, j’ai pu visualiser des résultats cohérents:



1. Représentation visuelle 2D en langage R sur 15 clusters

La plus grande difficulté de cette implémentation a été l’enchaînement des nouvelles structures de données/classes (Listes, Tuples, Arrays et Vector2). En tout cas, ceci représenterait un bon entraînement pour l’algorithme qui l’a suivi, le clustering des trajectoires en 3D.

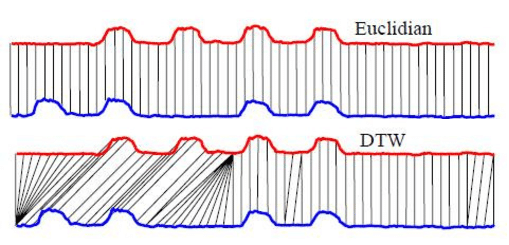
1. **L’algorithme DTW**

Il s’agit d’un nombre de trajectoires qu’on souhaite organiser en un nombre arbitraire de catégories selon leurs positions. Ceci peut aider à la visualisation des tendances ou des modèles, ce qui représente un outil important dans des domaines comme la médecine.

Pour arriver à ce but, on avait envisagé un passage de points 2D vers des trajectoires de points 3D. Avec l’implémentation déjà existante en tête, on a constaté le manque d’informations sur la distance entre 2 trajectoires, comme l’algorithme de base crée des clusters basés sur des distances minimales.

Après une recherche sur différents sites et articles scientifiques, la méthode la plus simple aurait été d’implémenter la distance Euclidienne, ce qui ne me donnerait pas de résultats assez précis. Avec elle, je calculerais une moyenne des distances entre chaque 2 points aux mêmes instants temporels sur les 2 trajectoires, ce qui ne nous donne qu’une approximation de la valeur réelle. La meilleure méthode et celle qu’on a enfin choisi a été l’algorithme DTW (Dynamic Time Warping).

Cet algorithme se base sur l’idée que 2 trajectoires peuvent se ressembler s ans que certains modèles apparaissent exactement aux mêmes instants temporels. On va comparer chaque point d’une trajectoire avec tous les points de l’autre pour trouver une ressemblance plus générale.



1. Comparaison entre le fonctionnement des algorithmes de calculs de distance[[7]](#ogr8bqkdjqls)

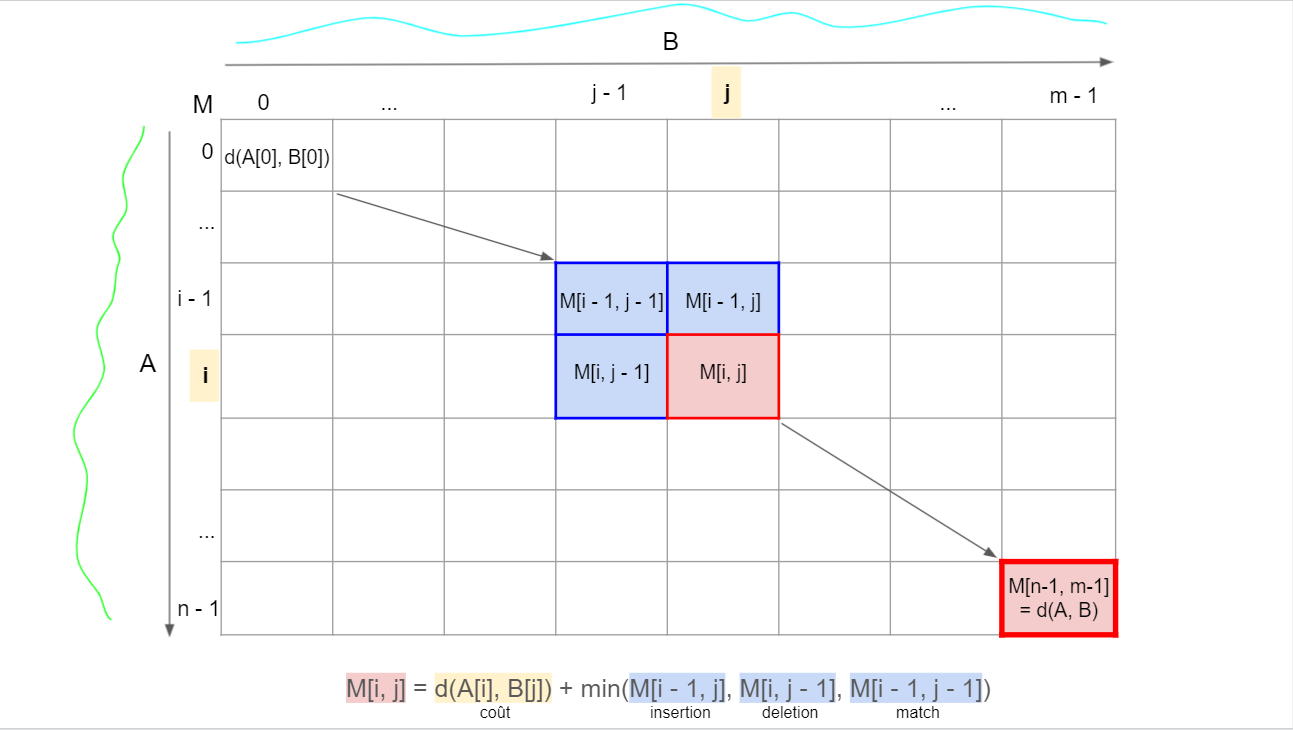
A ce stade-là, le plus difficile a été de trouver de bonnes sources qui ne soient ni trop complexes avec des notions mathématiques trop compliquées, ni trop simples et généraux avec des informations nécessaires leur manquant.[[8]](#m8dvx8gmuny9) A l’aide de plusieurs vidéos, sites web et d’une connaissance chercheur de mon tuteur de stage, on a réussi à confirmer quelle est la bonne étape pour s’arrêter, parce que l’algorithme pourrait continuer jusqu’au calcul d’une trajectoire moyenne.

On prend 2 trajectoires A et B en entrée (de longueurs n et m respectivement) et à partir d’elles, il faut calculer une matrice M de taille n x m selon les prochaines règles:

* M[0, 0] = d(A[0], B[0]) → L’origine est la distance entre les 2 points aux instants initiaux
* M[i, 0] = d(A[i], B[0]) + M[i - 1, 0]
* M[0, j] = d(A[0], B[j]) + M[0, j - 1]
* Les M[i, j] restants sont calculés à l’aide de la formule généralisée:

M[i, j] = d(A[i], B[j]) + min(M[i - 1, j], M[i, j - 1], M[i - 1, j - 1])

⇒ d(A, B) = M[n - 1, m - 1]



1. Illustration du fonctionnement de l’algorithme DTW
2. **Le passage vers k-means++**

Avec cette partie du code fonctionnelle, j’ai recréé les classes et les fonctions symétriquement par rapport à celles de k-means des points 2D:



**public class TCluster**

{

**public** Vector3[] Centroid;

**public** List<Vector3[]> Trajectories =

**new** List<Vector3[]>();

}

Avec cette implémentation, on considère qu’un centroïde est une trajectoire, donc un tableau de Vector3, et que les trajectoires contenues dans un cluster sont mises dans une Liste.

Dans une classe Trajectory, on factorise le code à l’aide des fonctions suivantes:



**public void kMeansTClustering**(**uint** k);

**public** Vector3[] **RandomWeightedCentroid**(Vector3[] Centroid, List<Vector3[]> Trajectories);

**public float** **DTWDistance**(Vector3[] A, Vector3[] B);

**public** List<Vector3[]> **ReadTrajectories**(**string** filename);

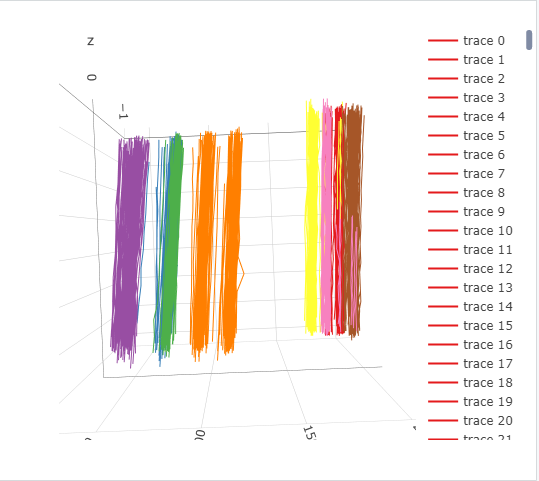
**public void WriteClusters**(List<TCluster> Clusters, **string** filename)

On commence comme avant avec un appel arbitraire dans une fonction implicite Start() vers kMeansTClustering. On lit les trajectoires depuis un fichier .csv avec le format “trajID,t,x,y,z” avec la fonction ReadTrajectories (cette fois-ci on la généralise pour lire un fichier donné en paramètre). L’algorithme se passe comme avant en remplaçant la distance entre 2 points 2D par la fonction DTWDistance, puis on écrit les données dans un fichier .csv avec le format “x,y,z,trajectory, cluster”.

La partie qui m’a demandé le plus d’effort pour déboguer a été l’initialisation des centroïdes. J’avais essayé de trouver une trajectoire médiane, mais ceci ne garantissait pas un choix uniforme et certains clusters n’apparaissaient dans aucun cluster. Pour résoudre cela, j’ai fait quelques changements pour passer de l’algorithme k-means vers la version optimisée k-means++[[9]](#sgrtbtjt35tf) qui emploie un choix aléatoire non-uniforme.

Il s’agit d’une fonction de probabilité avec des poids représentés par les distances de telle façon que les trajectoires les plus éloignées du centroïde courant ont plus de probabilité de devenir les prochains centroïdes que celles les plus proches. Après une courte recherche, j’ai trouvé une version simpliste de cet algorithme[[10]](#75nbbg50z4v4) que j’ai implémentée dans la fonction RandomWeightedCentroid.

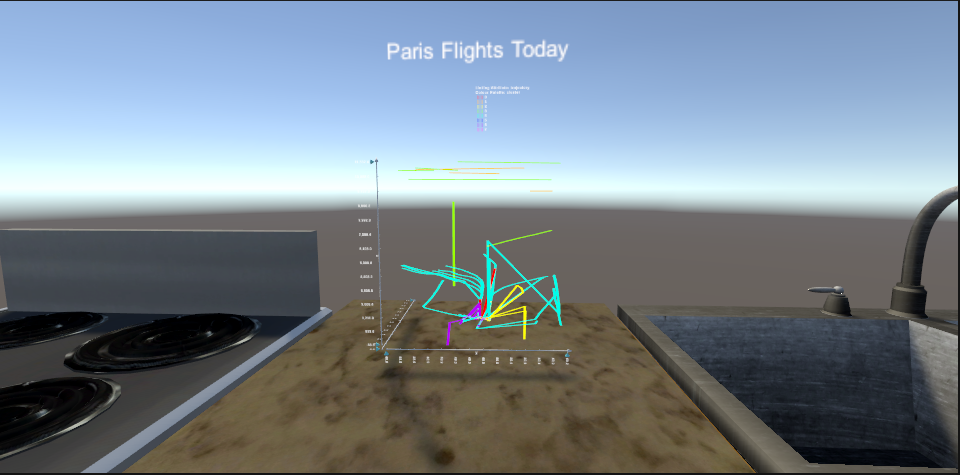
La visualisation a été rendue possible grâce à un script R externe et à d’autres fichiers pré-fabriqués trouvés sur des bibliothèques de méga-données en ligne. Les résultats ont été assez cohérents, alors on a choisi de passer vers une visualisation particulière en Unity pour automatiser le processus afin d’aller d’un fichier .csv en entrée vers une visualisation en sortie.

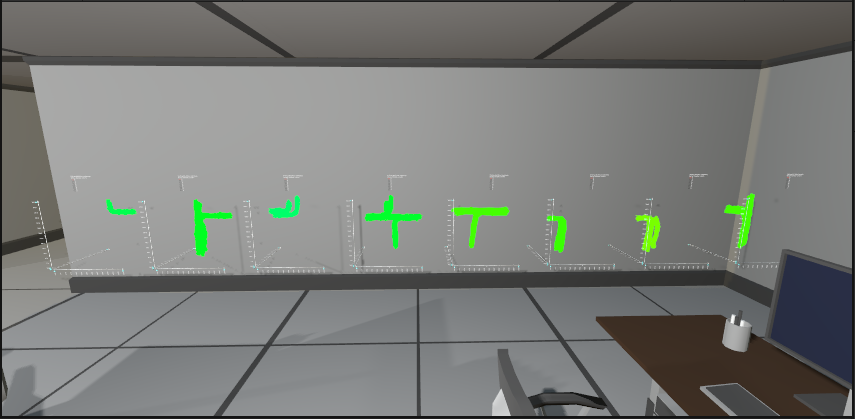


1. Représentation visuelle 3D en langage R sur 8 clusters
2. **La visualisation en IATK et la démo Unity**

Pour visualiser des clusters de trajectoires dans un environnement virtuel, on a décidé d’utiliser une bibliothèque créée par des chercheurs de l’Université Monash qui s’appelle IATK (Immersive Analysis ToolKit)[[11]](#nxvyrpiy69v4). Après l’installation dans Unity et le passage de k-means dans le projet IATK, j’ai créé 2 scènes Unity: une qui génère le fichier .csv à l’aide de k-means et une démo pour montrer ce qu’IATK pourrait faire dans un monde où la technologie AR est omniprésente.

Après une visualisation dans un contexte quotidien, on a décidé de créer une démo dans un milieu professionnel:



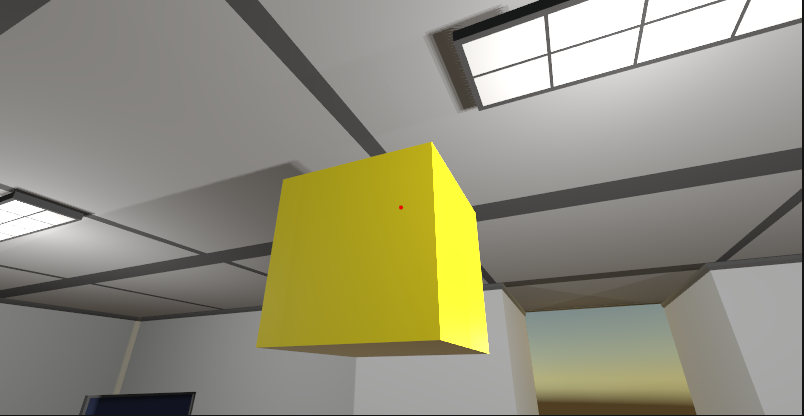


1. Scènes démo quotidien et professionnel IATK

Cette partie du stage m’a aidé à apprendre comment manipuler des objets 3D dans différentes scènes, comment les activer et désactiver, leur attacher des scripts, les organiser hiérarchiquement avec des résultats en chaîne (par exemple, bouger un objet translate aussi indirectement ses objets-enfant).

Pour rendre la démo interactive, j’ai trouvé sur Internet deux scripts[[12]](#63n77z3z7ysy)[[13]](#6jxnel42ihjn) que j’ai adaptés dans un seul script “SmoothMouseLook” pour bouger la caméra à l’aide des touches ZQSD et pour effectuer des rotations à l’aide de la souris.

Enfin, pour rendre la démo plus interactive, j’ai trouvé un script “SelectionManager”[[14]](#itx5rsob5bz7) qui peut sélectionner des objets à l’aide de l’API Physics.Raycast. Au début, certains objets avec l’étiquette “Selectable” pouvaient être sélectionnés pour changer la couleur et pour revenir à leur couleur initiale après.



1. Raycast rend le cube “Selectable” jaune quand la cible rouge est mise sur lui

Après avoir testé cette façon d’ interagir, je l’ai adaptée en mettant des cubes contenant chaque graphique IATK. Quand on met la cible sur eux, tant qu’on clique gauche, on le sélectionne, et si on relâche, on fait la désélection. Quand on sélectionne un objet, on désactive le script qui bouge la caméra et on l’active sur cet objet pour le contrôler tandis que la caméra reste sur place. Si on clique droit, le cube disparaît et nous montre le graphique. Quand on relâche, le cube réapparaît pour une nouvelle manipulation.

Comme il ne s’agit plus d’un projet statique, je mets un lien vers une courte vidéo YouTube pour montrer les résultats de tous ces systèmes ensemble:

<https://youtu.be/Ba3jaqK19S4>

1. **Conclusion**

Enfin, ce stage m’a donné l’occasion de commencer à coder des jeux vidéo à l’aide de la POO en C# et d’Unity. J’ai également utilisé d’autres outils, bibliothèques et documentations pour apprendre et pour appliquer des notions. Même si je change mon parcours professionnel, je suis sûr du fait que ces acquis seront bien utiles pour l’avenir.

J’ai réussi à mettre en place un algorithme à l’aide des sources web, je l’ai adaptée selon mes besoins, je l’ai généralisé et factorisé plusieurs fois pour arriver à une meilleure forme. J’ai utilisé un script externe dans un langage inconnu, j’ai appris à coder en C# et à attacher mes scripts à des objets, j’ai appris à manipuler des objets 3D dans une scène Unity, à installer des prérequis techniques et à enrichir des systèmes pour rendre un exécutable plus interactif afin de donner une démo de mon travail comme preuve définitive de mon progrès. Pendant les moments les plus difficiles, j’ai aussi appris à mieux me documenter, à mieux déboguer et à prendre mon temps pour trouver des alternatives aux solutions envisagées initialement.

J’ai aussi eu des échecs comme le sujet initial du stage raté à cause du manque d’appareils physiques et de SDK difficiles à installer qui auraient déjà été prêts pour moi si mon stage pouvait se passer en présentiel. Si j’avais eu plus de temps à disposition et plus d’expérience, j’aurais aussi pu optimiser mon code et l’améliorer pour trouver des clusters mieux répartis selon le choix des centroïdes. En plus, s’il y avait une bonne documentation d’IATK j’aurais bien pu l’utiliser pour aussi automatiser la génération des graphiques à l’aide du code au lieu de les placer et de les ajuster manuellement dans la scène.

Ce stage m’a donné un aperçu du travail dans un game engine et dans un milieu professionnel. J’ai eu des objectifs clairs, mais qui m’ont demandé beaucoup de recherches en plus pour mieux comprendre les concepts. Alors, j’ai appris à mieux penser à mon code et à demander de l’aide d’une façon claire et concise à des sources fiables comme les autres membres de l’équipe et certains forums. J’ai dû travailler avec des formats de fichiers spécifiques (comme le .csv), de nouveaux logiciels (Unity, RStudio), de nouvelles bibliothèques (IATK), des scripts externes (SmoothMouseLook, SelectionManager et les scripts R) et en utilisant git.

Pour conclure, je me sens plus capable de travailler en autonomie et de faire des recherches plus fiables, je sais comment bien factoriser et réorganiser mon code selon les besoins et comment adapter les mises en pratique des demandes selon mes savoirs courants. Je suis très content de mon choix pour ce stage comme j’ai acquis des connaissances qui m’aideront bien commencer mon parcours dans le développement des jeux vidéo, mais aussi pour d’autres sous-domaines de l’informatique.

1. **Bibliographie / Sources photo**

[1]<https://www.inria.fr/fr/centre-inria-bordeaux-sud-ouest>

[2] <https://learn.unity.com/project/roll-a-ball>

[3] <https://openclassrooms.com/fr/courses/6173501-debutez-la-programmation-avec-java>

[4] <https://catlikecoding.com/unity/tutorials/tower-defense/>

[5] <https://en.wikipedia.org/wiki/K-means_clustering>

[6] <http://cs.joensuu.fi/sipu/datasets/>

[7] <https://www.researchgate.net/figure/Euclidean-distance-vs-DTW_fig7_223966685>

[8] <https://riptutorial.com/algorithm/example/24981/introduction-to-dynamic-time-warping>

[9] <https://www.geeksforgeeks.org/ml-k-means-algorithm/>

[10] <https://stackoverflow.com/questions/1761626/weighted-random-numbers>

[11] <https://github.com/MaximeCordeil/IATK>

[12] <https://answers.unity.com/questions/172338/smooth-mouse-look.html>

[13] <https://forum.unity.com/threads/rotate-the-camera-around-the-object.47353/>

[14] <https://www.youtube.com/watch?v=_yf5vzZ2sYE>