|  |
| --- |
| https://s2.qwant.com/thumbr/0x0/5/a/8048bd0e3909c0d61fa4586efbcd99/b_1_q_0_p_0.jpg?u=http%3A%2F%2Fdircom.univ-amu.fr%2Fsites%2Fdircom.univ-amu.fr%2Ffiles%2Flogo_polytech.png&q=0&b=1&p=0&a=1 |
| Fitts’ Law sur téléphone portable |
| Rapport du Projet |
|  |
| **Clément STAMEGNA / Guillaume MICHON** |
| **17/04/2018** |

|  |
| --- |
| Rapport de fin de projet ReVa : Fitts’ Law sur téléphone portable. |

[1) Le Sujet 3](#_Toc514763176)

[1.1 Définition 3](#_Toc514763177)

[1.2 Présentation 4](#_Toc514763178)

[2) L’Organisation 6](#_Toc514763179)

[2.1 Mise en évidence des tâches et fonctions 6](#_Toc514763180)

[2.1.1 - la zone de jeu 6](#_Toc514763181)

[2.1.2 - Les Cibles et les interactions 6](#_Toc514763182)

[2.1.3 - Les Calculs et l’analyse 7](#_Toc514763183)

[2.1.4 - La sauvegarde et le transfert des résultats 7](#_Toc514763184)

[2.2 Priorisation des tâches 8](#_Toc514763185)

[2.3 Dépendances inter-tâches 8](#_Toc514763186)

[2.4 Planning 9](#_Toc514763187)

[2.5 Les Sprints 11](#_Toc514763188)

[2.5.1 - Sprint 1 : Core partie 1 11](#_Toc514763189)

[2.5.2 - Sprint 2 : Core partie 2 11](#_Toc514763190)

[2.5.3 - Sprint 3 : Core partie 3 11](#_Toc514763191)

[2.5.4 - Sprint 4 : taches secondaires 12](#_Toc514763192)

[3) Les Difficultés & Solutions apportées 13](#_Toc514763193)

[3.1 Sprint 1 : Core partie 1 13](#_Toc514763194)

[3.2 Sprint 2 : Core partie 2 13](#_Toc514763195)

[3.3 Sprint 3 : Core partie 3 15](#_Toc514763196)

[3.4 Sprint 4 : taches secondaires 17](#_Toc514763197)

[Conclusion 21](#_Toc514763198)

1) Le Sujet

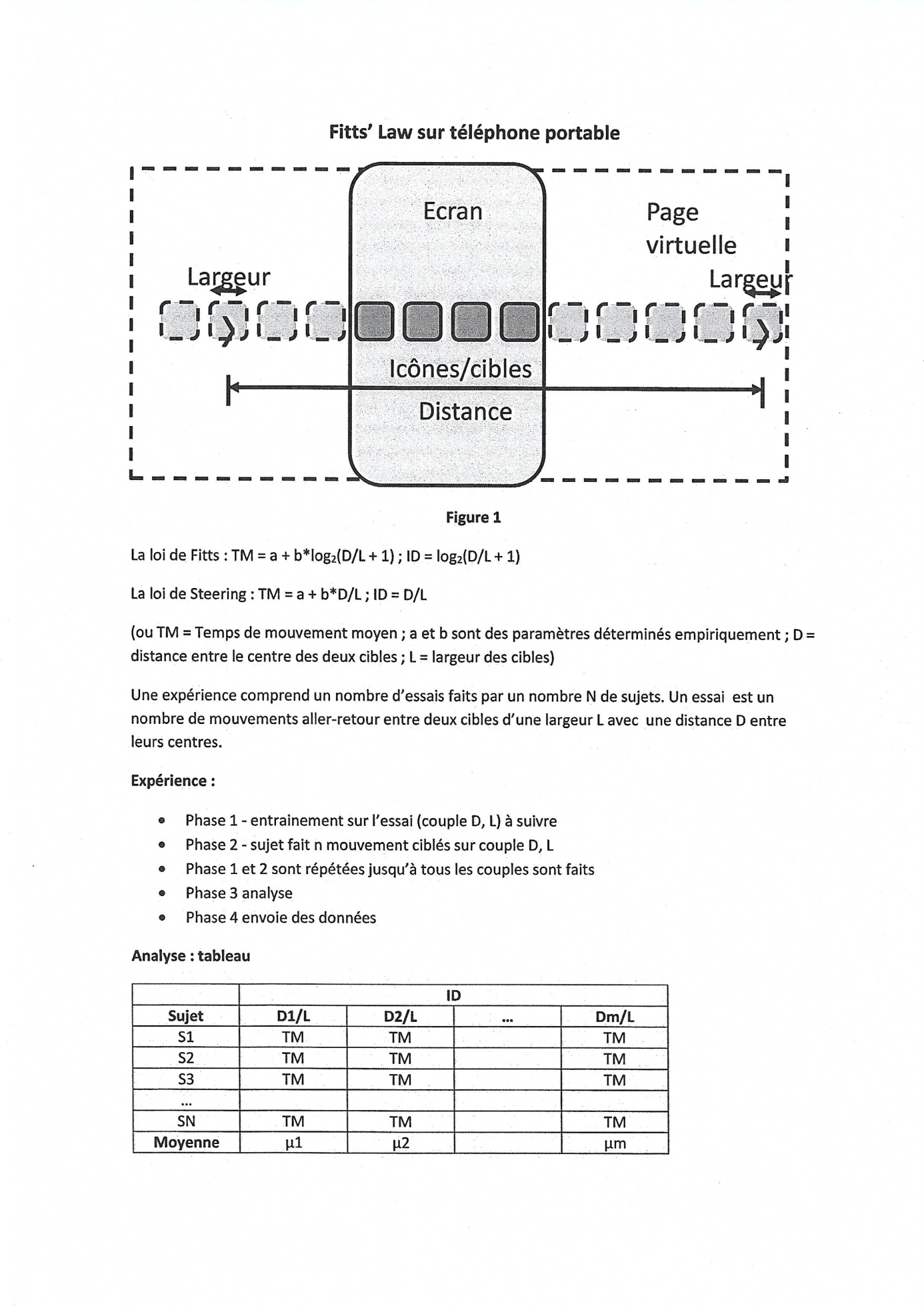
# 1.1 Définition

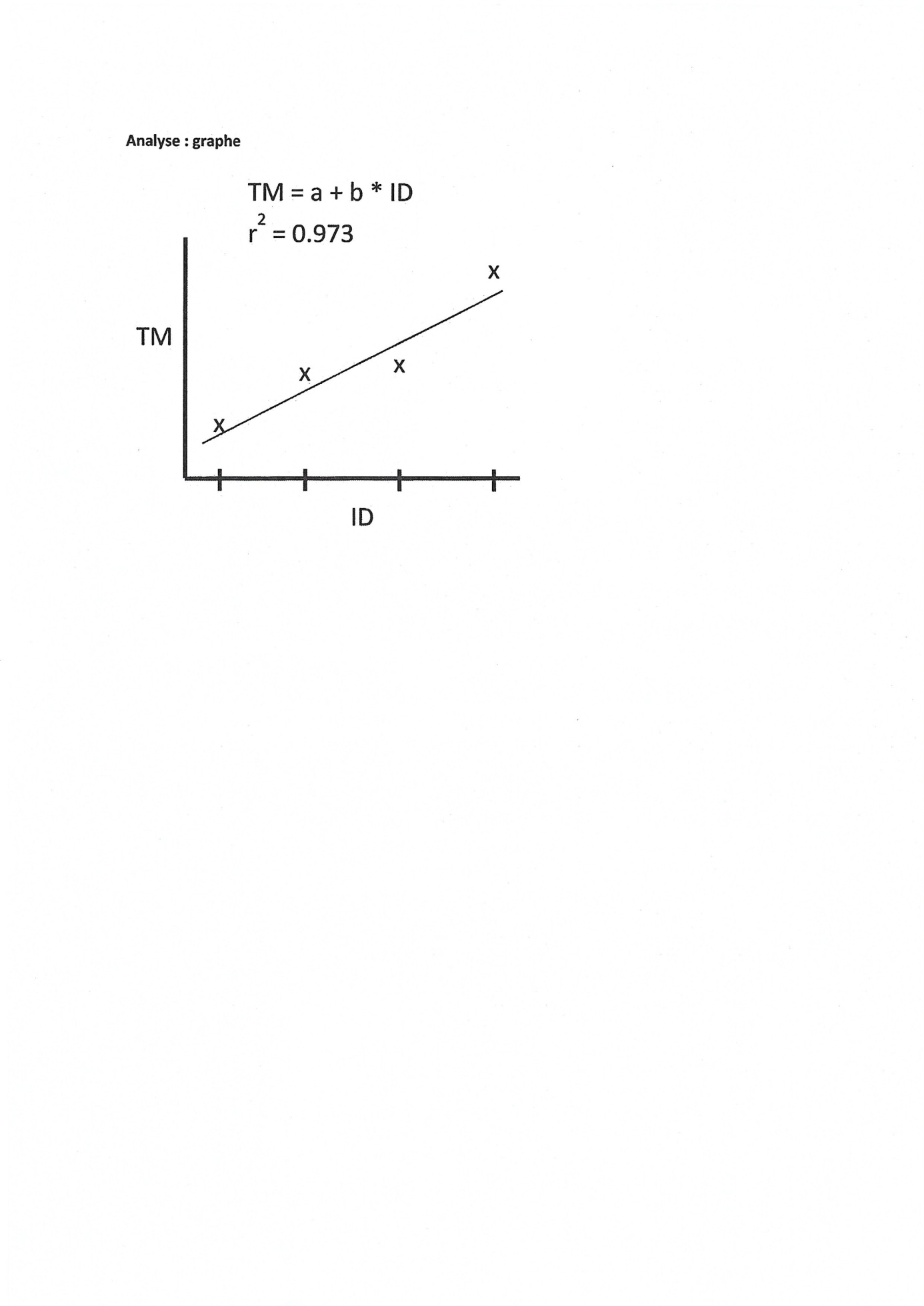
En psychologie expérimentale, en ergonomie et en interaction Homme-machine, la loi de Fitts est un modèle du mouvement humain qui prend en compte un indice de la difficulté d'une tâche. La formulation la plus courante actuelle exprime le temps requis pour aller rapidement d'une position de départ à une zone finale de destination, en fonction de la distance à la cible et de la taille de la cible. La loi de Fitts est utilisée pour modéliser l'acte de « pointer », à la fois dans le vrai monde, par exemple avec une main ou un doigt, et sur les ordinateurs, par exemple avec une souris.

Publiée par Paul Fitts en 1954, elle ne s'applique pas nécessairement aux interfaces tactiles actuelles

Source : https://fr.wikipedia.org/wiki/Loi\_de\_Fitts

# 1.2 Présentation





2) L’Organisation

# 2.1 Mise en évidence des tâches et fonctions

Lors de notre réunion d’explication du sujet, nous avons pu mettre en évidence plusieurs tâches principales et secondaires.

Nous avons décidé de les classer et de les regrouper en plusieurs parties.

## 2.1.1 - la zone de jeu

La zone de jeu est composée d’une « Page virtuelle », l’écran du téléphone permet d’avoir un aperçu de cette page.

Donc les tâches que nous devrons réaliser sont :

- La création de cette « Page virtuelle »

- La possibilité de déplacer notre écran sur cette page, le « Slide »

## 2.1.2 - Les Cibles et les interactions

Une fois la zone de jeu réalisée, nous avons besoin des cibles nécessaires à l’expérience. Ces deux cibles devront êtres placées sur la zone de jeu à une certaine distance avec une largeur fixe et nous devons pouvoir interagir avec.

Nous devons donc :

- Créer les Cibles

- Positionner les cibles sur la zone de jeu

- Mettre en place les interactions entre elles.

Les interactions sont définies comme suit :

- Si le joueur touche la cible il y a un « Hit »

- Si le joueur rate la cible il y a un « Miss »

- Si le joueur rate la cible mais glisse dessus cela reste un « Miss »

- Si le joueur touche la cible mais glisse en dehors cela compte comme un « Miss ».

## 2.1.3 - Les Calculs et l’analyse

Enfin nous devons utiliser les cibles et la zone de jeu précédemment construites pour effectuer les calculs de temps nécessaires pour l’expérimentation.

Pour cela nous devons mettre en place plusieurs mécanismes :

- Un « timer » pour calculer le temps

- Des structures de données utiles pour les calculs et les résultats

- Des Fonctions pour effectuer les calculs et analyses, notamment les TM (Temps Moyens), les moyennes des TM de tous les essais, les calculs de la Loi de Fitts, les calculs de la Loi de Steering et les comparaisons...

- Un mode d’administration pour paramétriser l’expérience avec le nombre d’essai pour chaque sujet ainsi que de fixer la distance entre les deux cibles.

- De nouvelles interactions sur les cibles pour l’expérience telles que le changement de distance entre les cibles, ou bien l’échec de l’essai si le joueur rate trop de cible.

## 2.1.4 - La sauvegarde et le transfert des résultats

Une fois les résultats obtenue il faut pouvoir les sauvegarder et les transféré pour des études ultérieures.

- il faut donc pouvoir utiliser un format pour organiser les données et écrire les fonctions associées pour la gestion de ces données.

Pour les Tâches optionnelles nous en avons défini quelques fonctions qui amélioreront l’expérience utilisateur telles que :

- La mise en place d’un menu

- L’apport de nouvelles règles de jeu, par exemple des cibles mouvantes, une zone de jeux en trois dimensions …

- Des graphismes améliorés

# 2.2 Priorisation des tâches

Maintenant que nous avons décrit toutes nos tâches, il faut prioriser les plus importantes et nécessaires au projet.

Suite à la réunion de présentation, nous avons séparé les tâches en tâches principales (cores) et tâches optionnelles.

Le Cœur du projet se situe dans les grandes parties 1, 2 et 3. Toutes les autres tâches sont optionnelles.

Ces parties seront donc traitées en priorité lors des sprints. Les tâches optionnelles seront traitées selon le temps restant pour le projet.

# 2.3 Dépendances inter-tâches

Les Parties principales possèdent des dépendances.

La partie 3 pour les calculs et l’analyse dépendent de la partie 2 les cibles et leurs interactions, en effet l’utilisation des cibles fait partie intégralement de l’expérience donc cette partie est nécessaire.

Elle doit donc être mise en place avant de commencer la partie 3, tous retards pris pour la partie 2 entrainent un retard sur la partie 3.

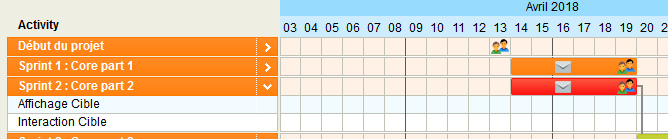
# 2.4 Planning

Nous avons insérer ces taches dans des sprints pour pouvoir réaliser un planning via un diagramme de Gantt.

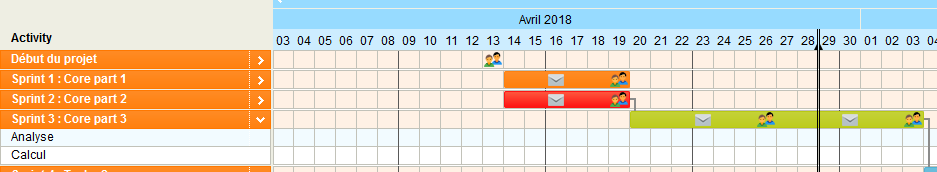
Partie 1



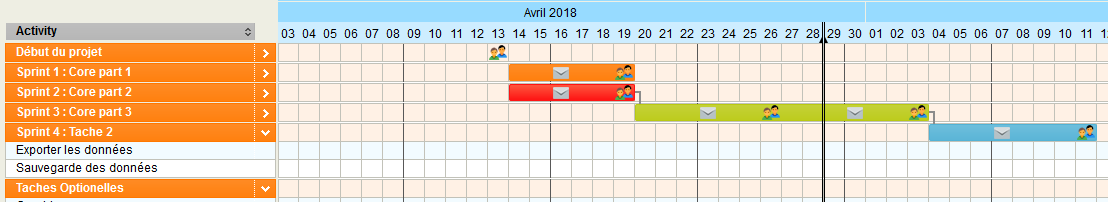
Partie 2



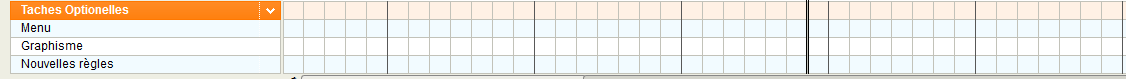
Partie 3



Partie 4



Partie 5



Chaque partie comporte son sprint et pour chaque sprint nous avons prévu un mail de spécification pour la démo de ce sprint (icône mail sur le diagramme) et la réunion pour ce sprint (icône deux petit bonhommes sur le diagramme), la réunion était utilisé pour montrer l’avancement via la démo, spécifier les prochaines étapes et voir si des composantes nécessitées des changements.

# 2.5 Les Sprints

## 2.5.1 - Sprint 1 : Core partie 1

Ce sprint était très simple, ce sprint nous a permit de nous familiariser avec le moteur Unity et son système de script en C# et de components.Pour notre objectif de sprint qui était la page virtuelle et la possibilité de slide, nous avons pu utiliser les composants de Panel et de « Scroll Rect » nous permettant d’implémenter ces fonctionnalités.

Pour la réunion de ce sprint, la démo présentée respecté bien les fonctionnalités requise, nous pouvons passer au sprint 2.

## 2.5.2 - Sprint 2 : Core partie 2

Pour ce sprint, le but était d’implémenter toute les fonctionnalités correspondante au cibles, leurs initialisation, leur interactions, pour cela nous avons commencé la programmation en C# de script unity pour notamment appliquer un comportement similaire au cibles et initialiser les zones de « miss ».

Cette fois si la réunion ainsi que la démo nous ont permis de spécifier plusieurs points sur les cibles et les zones de « miss ».

- si on glisse de la cible dans la zone de « miss » on considère cela comme un « miss ».

- nous devons utiliser un élément permettant de savoir si l’utilisateur à réussi ou a échouer a toucher la cible (changement de couleur de la cible, coloration de la cible).

Une fois ces points travaillés, nous passons au sprint 3.

## 2.5.3 - Sprint 3 : Core partie 3

Voici un sprint très important qui va permettre d’effectuer l’expérience. Pour ce faire nous allons créer un objet « Expérience » qui va contenir toutes les informations nécessaires.

- le nombre d’essai pour chaque participant de l’expérience,

- le nombre de mouvement à effectuer pour chaque essai.

- la liste des distances pour les essais.

- la liste de temps pour l’utilisateur actuelle

- le numéro de l’essai actuel

- ainsi que le tableau pour faire correspondre chaque sujet à sa liste de temps réalisée sur chaque essaies.

Il faut pourvoir paramétriser certains de ces attributs, pour cela nous implémentons un menu via le changement de scène d’unity « scene manager » et nous créons un nouvel écran ou l’utilisateur va pouvoir paramétrer plusieurs aspects tels que le nombre d’essai, la distance pour chaque essai et le nombre de mouvement a effectué pour réussir.

Il faut aussi changer quelques points dans la partie jeux pour intégrer l’expérience, il faut ajouter et modifier plusieurs interactions pour les cibles.

- permettre le changement d’essai

- créer le « timer », et enregistrer le temps de l’utilisateur à la fin de chaque essai

- lors de l’essai final, enregistrer tous les temps dans l’expérience et demander le nom de l’utilisateur.

- créer les conditions de réussite ou d’échec de l’essai.

La réunion et la démo ont permis de déterminer plusieurs aspects à travailler.

- quelques points IHM (interface homme machine) telle que l’ajout d’une « barre » de représentation pour mieux se repérer dans la page virtuelle, le changement de taille automatique des zone de miss…

- pour la prochaine partie prévoir une méthode d’exportation sous plusieurs formats (csv et json notamment).

## 2.5.4 - Sprint 4 : taches secondaires

Une fois les expériences réalisées, il faut pouvoir exporter les résultats obtenus pour cela nous créons une autre page permettant l’affichage et l’exportation des résultats.

Nous créons aussi des méthodes de conversion en fichier csv et json.

3) Les Difficultés & Solutions apportées

Nous allons maintenant revenir sur les Difficultés rencontrées lors de nos sprints, et quelles solutions nous avons pus apportées pour les résoudre.

## 3.1 Sprint 1 : Core partie 1

Ce sprint ne nous a pas posé de problème particulier, nous avons simplement appris les bases d’unity.

## 3.2 Sprint 2 : Core partie 2

Notre plus gros problème pour cette partie est au niveau de la composante des scripts unity, en effet un script doit être attaché à un ou plusieurs composants pour lui apporté des comportements d’initialisation, à chaque frame, …

Cependant si on exécute des méthodes complexe via un script et que ce script est aussi utile car il initialise des composant similaire, alors nous arrivons à plusieurs problèmes.

- puisque le script est attaché à plusieurs composants, les méthodes peuvent être appelées plusieurs fois.

- plusieurs références peuvent être faites pour le même objet donc, on risque par exemple de supprimer ou modifier un objet qui est utilisé par un autre composant, cela donne des erreurs d’objet non référencé, ou non initialisé.

- il faut aussi bien maitriser les initialisations via nos scripts, certaines parties peuvent nécessitent des données de composants qui n’existe pas, on arrive donc a des erreurs d’initialisation.

Durant ce sprint, nous avons eu des erreurs comme celle-ci, notamment :

- puisque le script qui gère les cibles est affilié aux deux cibles, certaines actions étaient effectuées en double et les variables qui servaient d’activation et de désactivation d’un coté n’étaient pas synchronisé entre les deux cibles.

Pour cela nous avons simplement permis à une seule cible d’éxécuter ces actions et pour les variables nous avons utilisé des variables « static » dons la mémoire est partagé entre toutes les instances d’un objet.

Public static boolean is\_active\_right ;

Public static boolean is\_active\_left ;

/\* les variables static sont désormais synchroniser entre les deux cibles \*/

…

…

…

Si (cible == « cible gauche ») {

Methode ()

/\* La Methode est maintenant lancer une seule fois, seul la cible gauche l’exécute \*/

}

…

…

…

- sur la fenêtre de jeux, les zones de « miss » autour des cibles nécessite de connaitre la position de leur cible attribué (cible droite ou cible gauche), or toutes les positions sont initialisé à peu près en même temps, donc les zone de « miss » ne trouvent pas les cibles qui ne sont pas encore initialisées. Cela provoque des erreurs.

Pour résoudre ce problème, nous utilisons le script d’initialisation des cibles pour lancer celui des zones de « miss ».

…

…

Void Start () {

…

… /\* Initialisation cible, toutes les données sont prêtes

…

Zone\_Miss\_Gauche.initialiser ()

Zone\_miss\_Droite.initialiser ()

/\* les zone de miss n’ont plus d’erreur de non initialisation car elles sont forcément après les cibles

}

…

## 3.3 Sprint 3 : Core partie 3

Pour cette partie nous avons décidé de rassembler toutes les données utiles pour l’expérience dans un objet « Experience ».

Le problème, l’objet doit être unique et de plus une fois initialisé, il ne doit pas être détruit durant tous le jeu.

Hors lorsqu’on change de scene, Unity remet à zero tous les objets et ne permet pas le passage normal d’objet entre les scènes.

Pour résoudre ce problème nous avons déjà cherché comment rendre l’objet unique. Nous pouvons utiliser le design pattern « Singleton ».

En génie logiciel, le **singleton** est un patron de conception (design pattern) dont l'objectif est de restreindre l'instanciation d'une classe à un seul objet (ou bien à quelques objets seulement). Il est utilisé lorsqu'on a besoin exactement d'un objet pour coordonner des opérations dans un système. Le modèle est parfois utilisé pour son efficacité, lorsque le système est plus rapide ou occupe moins de mémoire avec peu d'objets qu'avec beaucoup d'objets similaires.

Source : <https://fr.wikipedia.org/wiki/Singleton_(patron_de_conception)>

Ensuite nous cherchons comment rendre cet objet consistant dans le jeu.

Unity fournit une méthode « DontDestroyOnLoad(gameObject); ». Cette méthode permet de rendre un objet consistant c'est-à-dire il va rester charger dans le jeu même si on change de scène.

On combine alors le design pattern et cette méthode pour créer notre objet on obtient donc :

public class Experience : MonoBehaviour

{

public static Experience control;

/\*\* Attributs for the expérience

\*

\* Number of essai

\* Number of Movement

\* List of distances

\* Dictionnary for (Name of subject, List of time)

\*

\* Id Essai actuelle

\* List des temps de l'utilisateur actuelle

\*

\* Liste des Moyenne de tous les utilisateurs

\*\*/

public int nEssais;

public int nMouvement;

public List<float> l\_distance;

public Dictionary<string, List<float>> d\_time;

public int essaiAct = 0;

public List<float> l\_temps;

public List<float> l\_mean;

/\*\*

\* #Brief : Singleton when game start create an consistant instance of this object

\*/

void Awake()

{

if (control == null)

{

DontDestroyOnLoad(gameObject);

control = this;

l\_temps = new List<float>();

d\_time = new Dictionary<string, List<float>>();

l\_distance = new List<float>();

}

else if (control != this)

{

Destroy(gameObject);

}

}

## 3.4 Sprint 4 : taches secondaires

Puisque nous utilisons des objets complexes, nous ne pouvons pas utiliser les méthodes d’unity pour convertir nos données dans des formats de type Json ou bien Csv.

Nous allons donc créer nos convertisseurs vers les types JSON et CSV, il faut donc connaitre la composition de tels fichiers.

Pour le format json :

Nous pouvons donc écrire notre convertisseur, nous nous basons sur des méthodes tostring () basique qui renvoi une chaine de caractère, notre idée, renvoyé la chaine de caractère qui une fois mis dans le fichier forme le fichier json.

Voici notre implémentation.

/\*\*

\* #Brief : Parse the Experience object to JSON format to put in JSON FILE

\*/

public string ToJson()

{

string json = "";

// Debut du JSON

json += "{" + "\n";

// Objet Expérience

json += "\"Experience\"" + ": " + "{" + "\n";

// Nombre Essais

json += "\"NombreEssais\"" + ": " + nEssais + "," + "\n";

// NombreAllerRetour

json += "\"NombreAllerRetour\"" + ": " + nMouvement + "," + "\n";

// ListeDistances

json += "\"ListeDistance\"" + ": " + "[";

foreach (float t in l\_distance)

{

json += t + ", ";

}

json = json.Remove(json.Length - 2);

json += "]" + "," + "\n";

// Dictionnaire <Nom, ListeTemps>

/\*\*

\* Objet Nom0

\* [T0, T1, T2, ...]

\* Object Nom1

\* [T0, T1, T2, ...]

\*/

json += "\"Dictionnaire\"" + ": " + "{" + "\n";

foreach(KeyValuePair<string, List<float>> item in d\_time)

{

json += "\"" + item.Key + "\"" + ": " + "[";

foreach(float time in item.Value)

{

json += time + ", ";

}

json = json.Remove(json.Length - 2);

json += "]" + "," + "\n";

}

json = json.Remove(json.Length - 2);

json += "\n";

json += "}" + "\n";

// Fin Experience objet

json += "}" + "\n";

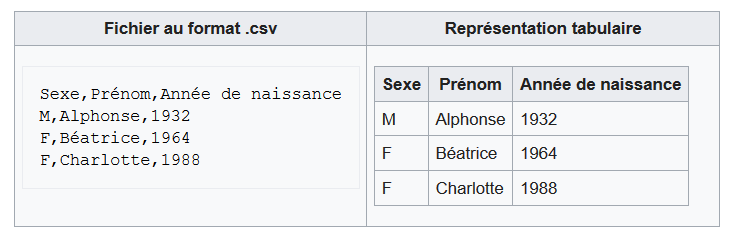
// Fin du JSON

json += "}";

return json;

}

Puis de Même pour le format Csv



Voici notre implémentation.

/\*\*

\* #Brief : Parse the Experience object to CSV format to put in CSV EXCEL FILE

\*/

public string ToCsv()

{

string csv = "";

// Objet Expérience

csv += "\"Experience\"" + "\n";

// Nombre Essais

csv += "\"NombreEssais\"" + ";" + nEssais + "\n";

// NombreAllerRetour

csv += "\"NombreAllerRetour\"" + ";" + nMouvement + "\n";

// ListeDistances

csv += "\"ListeDistance\"";

foreach (float t in l\_distance)

{

csv += ";" + t;

}

csv += "\n";

// Dictionnaire <Nom, ListeTemps>

/\*\*

\* Objet Nom0

\* [T0, T1, T2, ...]

\* Object Nom1

\* [T0, T1, T2, ...]

\*/

csv += "\"Dictionnaire\"" + "\n";

foreach (KeyValuePair<string, List<float>> item in d\_time)

{

csv += "\"" + item.Key + "\"";

foreach (float time in item.Value)

{

csv += ";" + time;

}

csv += "\n";

}

csv += "\n";

// Fin du CSV

return csv;

}

Conclusion