27 avril 2021

RAPPORT FINAL

[Projet Informatique Individuel]

PERRIER ALBAN

BORDEAUX INP

ENSC 2A – Groupe 1

SOMMAIRE

CODE COULEUR

nomDeFonction() → bleu (#0000ff)

nomDeScript.py → vert foncé 2 (#38761d)

nomDeDossier → fruits rouges (#980000)

.nomExtension → violet (#9900ff)

nomDeLibrairie → magenta (#ff00ff)

nomDeChamp ou nomDeVariable → orange foncé 2 (#b45f06)

GLOSSAIRE

(1) Librairie : En informatique, une bibliothèque logicielle est une collection de routines, qui peuvent être déjà compilées et prêtes à être utilisées par des programmes. https://fr.wikipedia.org/wiki/Biblioth%C3%A8que_logicielle

(2) Parser : Parcourir le contenu d'un texte ou d'un fichier en l'analysant pour vérifier sa syntaxe ou en extraire des éléments. https://fr.wiktionary.org/wiki/parser

(3) git : Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2. https://fr.wikipedia.org/wiki/Git

(4) json : JavaScript Object Notation (**JSON**) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple. https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

(5) csv : Comma-separated values, connu sous le sigle **CSV**, est un format texte ouvert représentant des données tabulaires sous forme de valeurs séparées par des virgules. https://fr.wikipedia.org/wiki/Comma-separated_values

I - INTRODUCTION

Présentation du projet

Dans le cadre de ce projet informatique individuel, j'ai fait le choix de contribuer au développement d'un outil de GazePlay (https://gazeplay.github.io/GazePlay/fr).
C'est un outil que ma sœur a utilisé lors de la réalisation de son mémoire de Master 2 de Neuropsychologie de l'enfant à l'Université Grenoble Alpes. Elle était donc en contact avec des membres de l'équipe s'occupant du développement de l'outil, et pour lequel ils accueillent avec plaisir des contributeurs de toute origine.
J'y ai donc vu l'opportunité d'allouer ce projet scolaire au développement d'un outil pleinement utile, notamment pour aider des enfants et adolescents porteurs de handicaps sévères. En effet, ces jeunes se voient privés d'outils et de jeux numériques en raison de leur difficulté, voire de leur incapacité à utiliser finement leurs membres (mains, pieds) ainsi qu'à développer le langage.
Il est en revanche possible d'utiliser leur regard grâce à un oculomètre (ou eyes-tracker en anglais), qui permet de détecter et d'enregistrer les mouvements des yeux et de suivre leur position sur l'écran. Avec des logiciels adaptés, cela leur offre l'opportunité d'interagir avec leur environnement. GazePlay fait partie de ces logiciels et propose des mini-jeux variés de complexité diverse, qui peuvent être proposés et adaptés en fonction du profil de l'enfant ou d'adolescent rencontré. Les utilisateurs peuvent à ce jour retrouver près de 60 jeux (labyrinthe, mémory, tarte à la crème, ...).

L'objectif de ces jeux est de permettre aux jeunes de se divertir, mais ils ont également une dimension de recherche et d'apprentissage. Ils permettent en effet d'entraîner la maîtrise du regard (fixation, balayage visuel, poursuite oculaire…) afin d'accéder à des interactions de plus en plus complexes. Après avoir discuté avec Didier Schwab, maître de conférences en informatique, membre du LIG (Laboratoire d'Informatique de Grenoble) et créateur de GazePlay, nous avons convenu que mon travail consisterait à développer un outil d'analyse du regard en "offline".

L'analyse du regard sur GazePlay n'est actuellement effectuée que lorsqu'un mini-jeu est terminé et se termine dès que l'écran est quitté. Or, des professionnels du corps médical souhaiteraient pouvoir analyser les données d'une partie de jeu sans avoir de contraintes logicielles et de temps.(préciser en deux mots en quoi cela est "un problème"?, expliquer pourquoi il y a ce besoin là en gros). À cette fin, j'ai utilisé le langage Python et une librairie adaptée à l'oculométrie.

Ressources utilisées

L'ensemble de mon code est disponible sur ce dépôt : https://github.com/aperrier004/PII
Le code est commenté en anglais afin de permettre une transmission de mon projet à un public plus large et qu'il serve ainsi de véritable contribution au projet GazePlay.

Pour l'ensemble de mon projet, les classes et fonctions sont codées sous Python 3.8 (https://www.python.org/), version stable me permettant d'utiliser correctement la librairie PyTrack (https://pytrack-ntu.readthedocs.io/en/latest/index.html) que j'ai choisi d'utiliser. J'ai utilisé Python, car il s'agit d'un langage que je ne connaissais pas, et sur lequel je souhaitais monter en compétences, puisqu'il est très utilisé et répandu dans tous les milieux.

Concernant l'interface graphique, j'utilise simplement la librairie Tkinter, très commune et exploitée? au sein de la communauté Python.
Les différents modules pour l'exécution de mon projet sont répertoriés dans le fichier "requirements.txt". J'ai également fait l'usage de l'IDE PyCharm (https://www.jetbrains.com/fr-fr/pycharm/), pour créer un environnement de travail virtuel.

Les données que j'utilise au sein de mon application proviennent des données générées par GazePlay à la fin de la réalisation d'un mini-jeu et qui sont stockées dans un fichier au format .json. Afin d'obtenir ces données, j'ai donc logiquement utilisé GazePlay, disponible à cette URL : https://gazeplay.github.io/GazePlay/fr.

II - PRÉSENTATION TECHNIQUE

Architecture

L'application est composée de différents dossiers et fichiers avec des responsabilités séparées.

Ainsi, on retrouve les dossiers :

- assets : contient les images de l'application
- GazePlay : contient des fichiers .json de GazePlay servant d'exemple d'utilisation de l'application, avec après utilisation de celle-ci, les fichiers .csv convertis créés ainsi que les dossiers Stimulus, Stimuli et Subjects (peuvent être vides)
- GESP : contient des projets GANTT pour les plannings prévisionnels et réels
- Livrables : contient tous les livrables du projet
- PyTrack : contient les classes pour utiliser la librairie PyTrack, certaines sont modifiées pour les besoins de mon projet et ne correspondent pas à la version de librairie officielle
- PyTrack examples : contient différents dossiers avec des données d'exemple pour l'utilisation de PyTrack et l'affichage graphique d'expériences avec des oculomètres de type smi, Tobii, et avec une architecture d'expérience particulière (NTU_Experiment)
- venv : contient un environnement virtuel d'exécution Python

Concernant les fichiers, on retrouve :

- main.py : fichier de lancement de l'application, il contient le fonctionnement général ainsi que l'interface graphique
- Parsing.py : contient les fonctions permettant le parsing et la conversion de données d'un fichier .json à un fichier .csv
- readingJSON.py : contient une fonction permettant de lire un fichier .json et d'en retourner les données qui nous intéressent
- README.md et README.en.md : Fichiers d'explications et d'installation du projet, en français et en anglais
- requirements.txt : contient les versions des packages à installer pour le fonctionnement du projet
- smiVisualizer.py : fichier qui m'a servi de tests et d'exemples pour d'abord l'utilisation de PyTrack, puis de l'utilisation de l'interface graphique Tkinter en suivant différents tutoriels
- statistiques.py : fichier contenant des fonctions permettant de récupérer des données sur le fichier .json

Choix techniques

Librairie pour l'analyse du regard

Le premier choix technique du projet fut de déterminer quelle technologie j'allais utiliser pour analyser le regard et quels outils j'allais pouvoir avoir à disposition.
M. Schwab avait préalablement posé quelques contraintes, la librairie devait être Open source, avec une licence qui en permettrait une libre utilisation . Elle devait par ailleurs disposer d'un certain nombre d'outils permettant la visualisation de données provenant d'un oculomètre. Le dernier critère concernait quant à lui l'activité de la librairie et de ses contributeurs. Ainsi, il était souhaitable d'avoir des ajouts réguliers et une communauté active.

Ainsi, j'ai pu établir un état des lieux des différentes librairies existantes en Python. J'ai notamment retrouvé Pygaze, librairie qui m'avait été recommandée. J'ai également trouvé Pyarbus, GazeParser et PyTrack.
J'ai souhaité essayer Pygaze afin de voir si elle pouvait répondre à mes besoins, mais j'ai rencontré des difficultés à l'installation et l'utilisation, notamment sur la version de Python (2.8) à utiliser qui était incompatible avec les packages que PyGaze utilisaient et qui eux n'était plus compatible avec cette version. Je me suis alors tourné vers la librairie avec la mise à jour la plus récente : PyTrack.
La librairie répondait à mes besoins et attentes avec ses trois outils de visualisation de données : une carte de chaleur, un graphique qui trace le parcours du regard et un troisième graphique qui examine le parcours du regard à travers le temps.

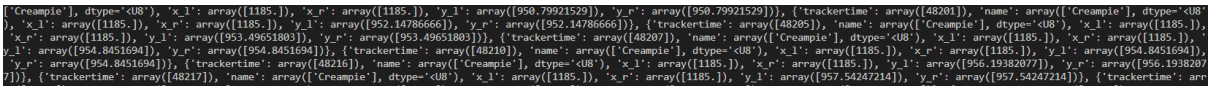
Modification de PyTrack

Lorsque j'ai souhaité mettre en pratique PyTrack avec des données d'exemples, j'ai rencontré certaines difficultés. Les fonctions de la librairie n'étaient pas forcément bien implémentées ou documentées et j'ai dû modifier le code source de la librairie afin de la rendre utilisable sur mon ordinateur.

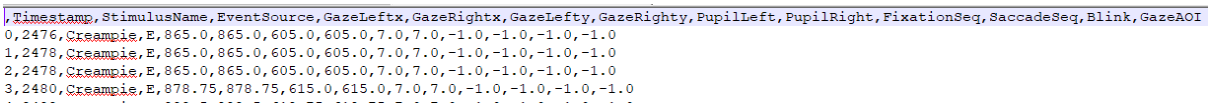
En conséquence, je ne travaille pas sur un clone de la dernière version de PyTrack, mais sur une version que j'ai modifiée (dans le dossier PyTrack directement). Ces modifications étaient nécessaires pour pouvoir réaliser la conversion de fichiers, mais également l'affichage, que je détaillerai ensuite.

Parsing et utilisation des données

Le second point technique du projet était d'utiliser un fichier .json provenant de GazePlay. Étant donné que les fonctions de PyTrack pour visualiser des données attendent en entrée un fichier .csv, j'ai donc dû "parser" ce fichier. La méthode consiste à parcourir et lire le fichier .json en extrayant des éléments qui nous intéressent pour les écrire dans un fichier .csv, dans un format syntaxique précis et attendu.



Exemple de données extraites du fichier .json de GazePlay



Exemple de lignes de mon fichier .csv avec des données parsées du fichier .json

J'ai ainsi créé la fonction *readJSON()* (script readingJSON.py) qui lit le fichier .json donné en entrée et retourne un tableau avec les données qui nous intéressent. Puis, dans la fonction *toBase()* (script Parsing.py), on associe chaque donnée à l'entête de la colonne du nouveau fichier .csv correspondante. Pour parcourir toutes les données du .json, on compte le nombre d'éléments dans le tableau 'coordinatesAndTimeStamp', et en le parcourant, on pourra par exemple récupérer pour chaque élément dans le champ 'time', l'équivalent de ce qui sera la valeur de chaque ligne pour la colonne 'Timestamp' dans le fichier .csv.

La structure des données correspond à ce qu'attendent les fonctions de PyTrack, j'ai dû en respecter les libellés et l'ordre des différentes colonnes.

Utilisation de Tkinter

Dans le but de réaliser une interface graphique, mon choix s'est porté sur la librairie Tkinter, qui me semblait la plus répandue pour l'utilisation de scripts Python.
Ainsi, j'ai pu réaliser quelques tutoriels, axés sur la représentation de graphiques sur des fenêtres et différentes pages, avec la librairie matplotlib (https://matplotlib.org/).
Après avoir schématisé l'interface et répertorié les éléments que je souhaitais obtenir, j'ai souhaité repartir d'un tutoriel qui avait une architecture qui me plaisait (avec une page de menu, puis différentes pages que l'on pouvait parcourir).
Cependant, en voulant insérer les graphiques voulus sur les différentes pages, je me suis rendu compte que les fonctions *gazeHeatMap()*, *gazePlot()* et *visualize()* (script Stimulus.py dans le dossier PyTrack) n'étaient pas adaptées car elles créaient chacune leur propre fenêtre d'application. J'ai alors fait le choix de les modifier pour qu'elles retournent seulement une figure, que je pourrais ensuite utiliser et afficher dans une figure (équivalent d'un canva) de ma propre application.

Interface graphique

Mon application avait donc pris forme et fonctionnait plutôt bien, mais elle n'était pas utilisable. Le chemin du fichier de données .json devait être écrit dans le code, ce qui n'est pas souhaitable. Ainsi, j'ai créé un bouton ouvrant l'explorateur de fichiers et permettant de sélectionner un fichier. Ce choix a eu pour conséquence sur mon application que le nom du fichier à utiliser ne soit transmis qu'après son lancement, mais aussi qu'il soit transmis aux autres pages seulement après la sélection du fichier. Ne sachant pas comment opérer, j'ai cherché plusieurs solutions pour passer des données d'un écran de mon application à un autre, avec succès concernant le nom du fichier. Cependant, je n'ai pas réussi concernant l'objet *Stimulus*, qui permet de créer les différents graphiques avec les fonctions de PyTrack. Je détaillerai davantage ce problème dans la partie difficultés.
À la suite de cela, j'ai fait le choix de reprendre intégralement mon interface, en ne créant que ce dont j'avais besoin, c'est-à-dire deux écrans. Grâce à cela, une fois la sélection du fichier effectuée sur le premier écran, je n'ai eu aucun mal à créer tous les objets et graphiques nécessaires sur un deuxième écran.

III - NOTICE

Installation

Installation de l'environnement de Python

Si vous ne possédez pas déjà un environnement Python sur votre ordinateur, vous pouvez :

- télécharger une version de Python (le projet fonctionne en version 3.8.1) : https://www.python.org/downloads/
- puis télécharger un IDE comme PyCharm à cette adresse : https://www.jetbrains.com/fr-fr/pycharm/download/#section=windows si vous ne souhaitez pas passer par l'installation en ligne de commande, ignorez le paragraphe suivant

Installation de l'application en ligne de commande

1. Clonez le projet à l'adresse suivante : https://github.com/aperrier004/PII (ou téléchargez le code puis décompressez l'archive)
2. Ouvrez un terminal et placez-vous dans le dossier du projet "PII" (avec la commande `cd`)
3. Tapez la commande suivante :

```
pip install -r requirements.txt
```

4. Lancez l'application avec la commande : `python main.py`
5. Vous arrivez sur le premier écran, sélectionnez un fichier .json (biboulеJump.json par exemple), puis cliquez sur le bouton "Show Graphs", une nouvelle fenêtre s'affichera et vous pourrez visualiser les données.

Installation de l'application avec un IDE

1. Clonez le projet à l'adresse suivante : https://github.com/aperrier004/PII (ou téléchargez le code puis décompressez l'archive)
2. Ouvrez votre IDE pour Python et ouvrez le projet en sélectionnant le dossier "PII"
3. Sélectionnez ou configurez un interpréteur Python compatible (version 3.8 par exemple)
4. Sélectionnez "requirements.txt" comme fichier de dépendances pour les installer
5. Attendez que toutes les dépendances se téléchargent et que le projet se charge avec ses composants, cela peut prendre quelques minutes
6. Ouvrez le fichier "main.py"
7. Cliquez sur l'onglet "Run" de la barre de menu, puis sur "Run"

Utiliser des données de GazePlay

Si vous êtes utilisateur du logiciel GazePlay (disponible ici : https://gazeplay.github.io/GazePlay/fr), vous pouvez utiliser vos propres fichiers de données .json directement en les sélectionnant dans l'application.
Vous pouvez en trouver avec un chemin d'installation par défaut ici :
C:/Utilisateurs/[NOM_UTILISATEUR]/GazePlay/statistics/[NOM_JEU]

Une vidéo de démonstration est disponible ici :
https://drive.google.com/file/d/1IFsiGV17QBc5SvR9485ndJc6O3Vig-gf/view?usp=sharing
Une vidéo de présentation du projet est disponible ici :
https://drive.google.com/file/d/1wwqeQf3lesuRvH2-Da0CwvH0jIyrVALz/view?usp=sharing

Ces informations sont également disponibles dans le fichier README.md du code source (également disponible en anglais).

Utilisation et fonctionnalités

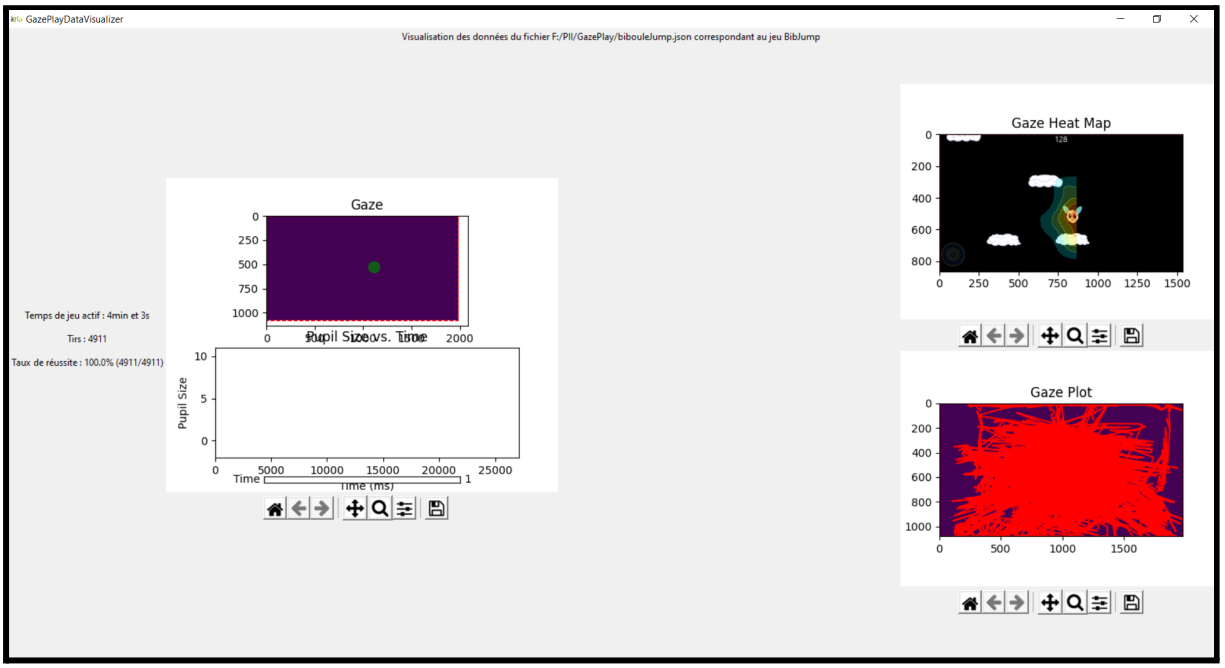
Au lancement de l'application, une fenêtre s'ouvrira. L'action a réaliser est de cliquer sur le premier bouton pour sélectionner un fichier .json provenant de GazePlay (par défaut, l'explorateur de fichier vous place dans le dossier de l'application contenant déjà des données d'exemples). Puis, une fois sélectionné, vous pouvez appuyer sur le second bouton pour faire apparaître les différents graphiques.

Premier écran de l'application

En cliquant sur ce bouton, si un fichier .csv avec le même nom que le fichier .json sélectionné n'existe pas, alors l'application va procéder à la création de ce nouveau fichier. Cette création de fichier peut être plutôt longue, il est possible d'en suivre la progression dans la console.

Il est à noter qu'il est possible d'ouvrir plusieurs fenêtres de visualisation de données en sélectionnant des fichiers différents et en cliquant sur le bouton de visualisation. Il est ainsi possible de visualiser plusieurs données en même temps.

Une fois le fichier .csv trouvé ou créé, alors l'application affiche une nouvelle fenêtre composée d'une carte de chaleur (représentation graphique de données statistiques qui fait correspondre à l'intensité du temps de regard sur une zone variable un nuancier de couleurs sur une matrice à deux dimensions), mais également un graphique du parcours du regard de l'utilisateur (tracé rouge). On retrouve un troisième graphique, à gauche, mais qui ne fonctionne pas encore correctement. Il est en effet censé afficher le tracé du regard au cours du temps.
Enfin, tout à gauche, on peut voir des statistiques sur la réalisation du jeu sur GazePlay.



Écran de visualisation de l'application

Pour chacun de ces graphiques, grâce aux barres d'outils en dessous, il est possible de :

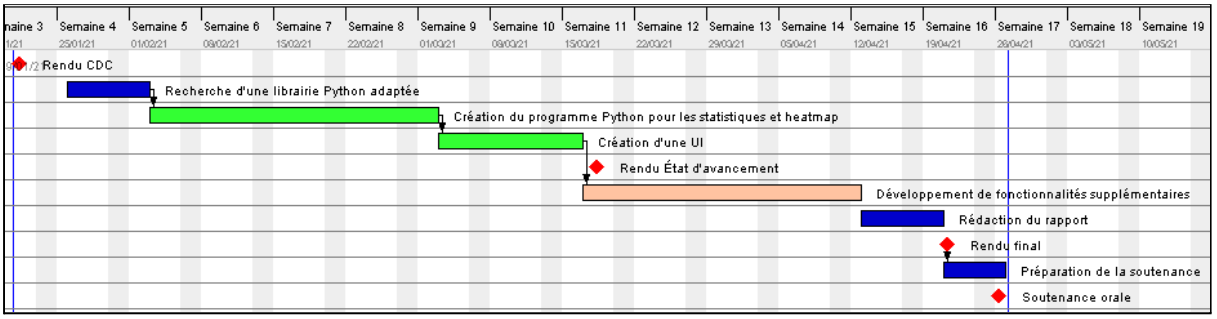
- Enregistrer l'image représentant le graphique
- Déplacer la région étudiée/visualisée des données du graphique selon l'échelle
- Zoomer sur une partie rectangulaire du graphique
- Configurer le graphique, placement de droite à gauche, haut en bas, ...

IV - GESTION DE PROJET

Planning prévisionnel

Mes objectifs et tâches réalisées au cours de ce projet ont été définis et mis à jour sur ce Trello (https://trello.com/b/FeoYRLDQ).

Voici un diagramme de GANTT représentant le planning prévisionnel de mon projet :

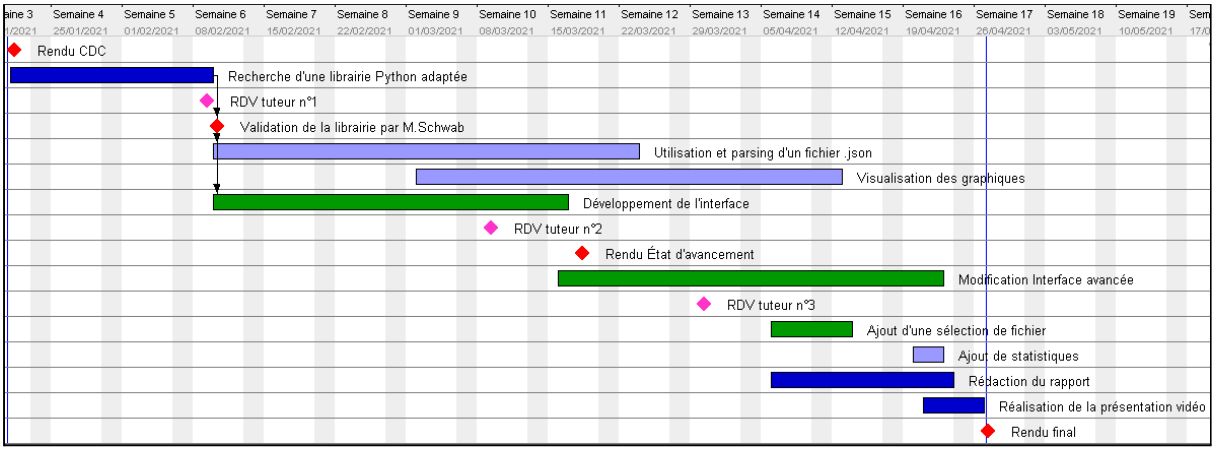


Listes des tâches associées :

Nom	Date de début	Date de fin
Rendu CDC	21/01/21	21/01/21
Recherche d'une librairie Python adaptée	26/01/21	02/02/21
Création du programme Python pour les statistiques et heatmap	03/02/21	02/03/21
Création d'une UI	03/03/21	16/03/21
Rendu État d'avancement	18/03/21	18/03/21
Développement de fonctionnalités supplémentaires	17/03/21	12/04/21
Rédaction du rapport	13/04/21	20/04/21
Rendu final	21/04/21	21/04/21
Préparation de la soutenance	21/04/21	26/04/21
Soutenance orale	26/04/21	26/04/21

Planning réel

Voici le planning réel de mon travail pour ce projet :



➤ Tâche de rédaction ou de recherche ➤ Tâche de développement de fonctions

➤ Tâche de développement d'interface

◊ Jalon de rendu de livrable ◊ Jalon de rdv avec le tuteur

Listes des tâches associées :

Nom	Date de début	Date de fin
Rendu CDC	21/01/2021	21/01/2021
Recherche d'une librairie Python adaptée	26/01/2021	09/02/2021
RDV tuteur n°1	09/02/2021	09/02/2021
Validation de la librairie par M.Schwab	10/02/2021	10/02/2021
Utilisation et parsing d'un fichier .json	10/02/2021	23/03/2021
Visualisation des graphiques	02/03/2021	12/04/2021
Développement de l'interface	10/02/2021	16/03/2021
RDV tuteur n°2	09/03/2021	09/03/2021
Rendu État d'avancement	18/03/2021	18/03/2021
Modification Interface avancée	16/03/2021	22/04/2021
RDV tuteur n°3	30/03/2021	30/03/2021
Ajout d'une sélection de fichier	06/04/2021	13/04/2021
Ajout de statistiques	20/04/2021	22/04/2021
Rédaction du rapport	06/04/2021	22/04/2021
Réalisation de la présentation vidéo	19/04/2021	23/04/2021
Rendu final	27/04/2021	27/04/2021

Écarts et difficultés

On peut noter plusieurs écarts entre mon planning prévisionnel et mon planning réel.

Utilisation de Python

J'ai en effet rapidement rencontré des difficultés que je ne pensais pas rencontrer. En effet, lorsque j'ai fait des recherches pour trouver différentes librairies d'exploitation de données pour afficher des graphiques, je souhaitais les essayer rapidement et voir celles qui me convenaient le mieux. Cependant, j'ai souvent eu des difficultés à mettre en place l'environnement Python nécessaire. N'ayant jamais utilisé Python auparavant, je ne comprenais souvent pas les notices d'installation et d'utilisation des librairies. De plus, je rencontrais souvent des problèmes de version de packages que je devais installer pour utiliser une librairie qui se basait finalement sur une ancienne version de ces packages, ce qui rendait impossible son utilisation. Ce problème, je l'ai rencontré fréquemment puisque la plupart des librairies intéressantes étaient plutôt datées en termes de dernières mises à jour, ce qui m'a appris qu'il est toujours préférable de regarder l'état du dépôt git d'une librairie et de regarder le nombre d'issues et l'activité récente de celle-ci.

Modification des fonctions

Une fois ma librairie déterminée, je me suis donc attelé à la création d'un script Python me permettant d'afficher des graphiques, ce que j'ai d'abord réalisé avec des données d'exemple fournies. L'étape suivante était donc d'exploiter des données réelles provenant de GazePlay. J'ai ainsi appris le principe de "parsing" et récupéré les données du fichier .json correctement pour pouvoir les transmettre aux fonctions de PyTrack pour créer le fichier .csv attendu. Or, les fonctions de PyTrack ne prennent pas en compte les fichiers .json, j'ai donc créé moi-même ces fonctions, pour pouvoir ensuite réutiliser celle de PyTrack.

Réalisation d'une interface graphique

Il s'agissait ensuite de créer une interface graphique pour l'utilisation réelle de l'application. Vraisemblablement, je n'avais pas anticipé le temps de formation sur la librairie Tkinter, qui est simple, mais sur laquelle j'ai rencontré des difficultés pour obtenir le résultat que je souhaitais. Tout d'abord, j'ai dû modifier les fonctions de la librairie PyTrack (comme expliqué dans mes choix techniques), puis j'ai passé beaucoup de temps à déterminer quel était le meilleur moyen pour moi de transmettre de façon correcte le fichier à analyser et les données à l'application. En effet, si le fait de mettre un bouton "parcourir" pour sélectionner le fichier à utiliser était évident, permettre la conversion en .csv et l'utilisation de celui-ci pour afficher les graphiques l'étaient moins. Je souhaitais avant tout une application permettant un parcours au sein d'une même fenêtre, avec différents écrans (comme il est possible de le voir dans l'application d'exemple smiVisualizer.py). Mais cela ne fonctionnait pas, car la création de l'objet Stimulus et la création des graphiques ne pouvait se faire qu'une fois le fichier .csv donné, or les écrans correspondant aux graphiques ne mettaient pas à jour leur affichage. J'ai essayé différentes méthodes, notamment la création d'événement qui s'appelait lorsqu'on affichait l'écran, mais sans succès. J'ai donc travaillé sur l'interface graphique pendant beaucoup de temps, et j'ai dû revoir cette organisation pour en arriver au résultat final.

Bilan sur le planning

Toutefois, ce temps que j'ai passé en plus pour travailler sur des points que j'avais sous-estimés en termes de charge de travail ne m'a pas pénalisé. L'objectif qui m'avait été fixé au départ du projet était d'avoir une application avec les fonctionnalités similaires à l'écran de fin de jeu de GazePlay, résultat auquel je suis parvenu. Je pensais atteindre cet objectif avant le rendu intermédiaire, et souhaitais ensuite consacrer le reste du temps de projet à développer des fonctionnalités supplémentaires. J'ai finalement consacré ce temps à la consolidation et l'amélioration des fonctionnalités de base.

V - PISTES D'AMÉLIORATION

Cette partie définit les parties du projet pouvant être retravaillées sur lesquelles j'ai rencontré des difficultés, ainsi que des perspectives d'évolution pour le projet.

Les fonctionnalités

Visualize

Le graphique à gauche de l'application est créé à l'aide de la fonction *visualize()* du script Stimulus.py de PyTrack. Actuellement, il n'affiche aucune donnée. J'ai longtemps essayé de chercher quelle en était la cause, car la fonction lit bien les données du fichier .csv, mais il y a un problème au moment d'ajouter des points selon une certaine échelle et la figure reste vide. L'amélioration consisterait donc à faire marcher cette fonctionnalité, qui s'avérerait très utile puisqu'il est tout à fait intéressant de visualiser le parcours du regard à travers le temps, point par point. Au départ, la fonction ne renvoyait même pas une figure et faisait planter l'application, mais au fur et à mesure de mes essais et modifications je suis tout de même parvenu au point ou la figure s'affiche, mais reste encore vide.

Nouveaux outils d'analyse

En dernier lieu, afin de faire évoluer ce projet, il sera requis d'ajouter de nouveaux outils d'analyse du regard. Cela serait possible en rajoutant l'utilisation d'une autre librairie (en réussissant à utiliser PyGaze par exemple), en s'affranchissant de l'environnement de travail requis au départ (cf. explication du choix de la librairie).

L'interface

L'interface graphique peut également être améliorée sur plusieurs points.

Affichage d'images

Actuellement, l'image correspondant au fichier de données du jeu n'est affichée que sur le graphique de carte de chaleur, alors qu'elle pourrait l'être également sur les graphiques de tracé du regard et de visualisation dans le temps. De plus, l'image est n'est actuellement affichée que pour les données d'exemple de l'application. En effet, pour l'afficher, il est nécessaire qu'elle soit placée au sein d'un dossier Stimuli, qui se trouverait au même endroit que le fichier .json sélectionné. De plus, il faut que l'image soit au format .jpg ou .jpeg, alors que sur GazePlay elles sont enregistrées au format .png. Ces détails sont modifiables en changeant le chemin de recherche d'image au sein des fonctions *gazeHeatMap()*, *gazePlot()* et *visualize()* du script Stimulus.py. Le résultat intermédiaire auquel je suis parvenu est d'afficher l'image dans les autres graphiques certes, mais elle n'était pas au bon format, dû notamment au changement d'échelle des figures.

Comparaison de données

En utilisation réelle de l'application, il pourrait être intéressant de pouvoir comparer plusieurs jeux de données entre eux. Ainsi, même s'il est actuellement possible d'ouvrir plusieurs fenêtres, il serait plus pratique et efficace de tout afficher sur un même écran. Il serait alors pertinent d'afficher sur les mêmes graphes différentes données, avec des couleurs différentes pour différencier les fichiers étudiés. Une autre possibilité serait de représenter et mettre en évidence sur un graphique les différences ou ressemblances, entre deux parties d'un même jeu par exemple.

Barre de progression

Pour le moment, lorsqu'un fichier .csv est créé suite à la sélection d'un fichier .json, sa création est plutôt lente. Cela a pour impact que l'on suppose que l'application a planté ou est au point mort, il faudrait alors pour l'utilisateur rajouter un écran de progression ou en tout cas une barre de progression pour lui indiquer que l'application est toujours en cours d'exécution et effectue une tâche en arrière-plan. Pour l'instant, on peut seulement visualiser la progression dans la console d'exécution de l'application, car je n'ai pas réussi à mettre en place cette barre de progression. J'avais essayé d'en créer une avec le widget *Progressbar* de Tkinter, en simulant une avancée "fausse", mais cela ralentissait l'exécution. L'idée serait donc plutôt de calculer et d'estimer le temps restant de conversion du fichier se termine d'être créé. Cela serait possible en chronométrant par exemple les cinq premières secondes de conversion du fichier .json, et en regardant quel pourcentage du fichier a été parcouru, ainsi, on pourrait calculer le temps restant. Je n'ai malheureusement pas eu le temps de mettre cela en place, notamment à cause de mon changement de logique d'interface.

VI - CONCLUSION

En conclusion, j'ai développé une application de visualisation de données provenant du logiciel GazePlay dans le cadre de mon projet informatique individuel de deuxième année, encadré par Pierre-Alexandre FAVIER, professeur et maître de conférences de l'ENSC, ainsi qu'aider par Didier SCHWAB, maître de conférences en informatique et Sébastien RIOU, tous deux membres du LIG (Laboratoire d'Informatique de Grenoble).
Lors de ces 3 mois de travail, j'ai pu acquérir une expérience d'analyse et de réalisation d'un programme informatique et ai également pu gérer individuellement un projet dans son intégralité.

Le développement de mon application a été très enrichissant pour moi, car il m'a permis de monter en compétences sur des technologies que je n'avais jamais utilisé auparavant. J'ai ainsi pu avoir une première expérience avec le langage Python, langage simple dans son fonctionnement, mais qui m'a demandé un temps de compréhension sur l'installation de l'environnement de travail et des différentes versions existantes. Pour réaliser ce projet, j'ai également utilisé les librairies PyTrack et Tkinter. J'ai vraiment pu manipuler et comprendre entièrement PyTrack. J'ai ainsi pu la modifier pour obtenir ce que je souhaitais et travailler avec des librairies en plus, telles que matplotlib, numpy, pandas et scipy. Concernant l'interface graphique, j'ai utilisé Tkinter, sur laquelle j'ai dû beaucoup travailler afin de réussir à obtenir le comportement et l'allure que je recherchais.

En somme, cette expérience m'a permis d'apprendre à mener un projet informatique de la conception à la livraison, et ce de manière individuelle. Cela m'a permis de prendre conscience des difficultés que l'on peut rencontrer lorsque l'on conduit seul un projet.

VII - ANNEXES

Le script gazeMenu.py correspond à la première interface graphique à laquelle j'avais réfléchie. Il est fonctionnel et il est possible de l'exécuter sans erreurs.

Le script gazeMenuBrowse.py correspond à mes essais infructueux d'interface graphique où le fichier .json pouvait être sélectionné directement au sein de l'application, j'y utilise notamment des variables partagées entre les écrans. Le script n'est pas fonctionnel.

Sources supplémentaires :

Logiciel et projet GazePlay : https://gazeplay.github.io/GazePlay/fr

Librairie PyTrack : https://pytrack-ntu.readthedocs.io/en/latest/

Librairie Tkinter : https://docs.python.org/fr/3/library/tkinter.html

Logiciel GanttProjet 3.0 : https://gantt-project.fr.uptodown.com/windows

Éditeur de code utilisé (VS Code) : https://code.visualstudio.com/