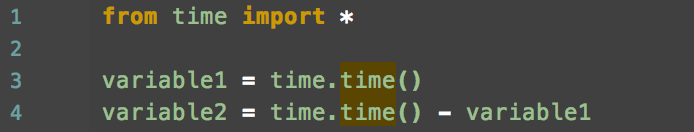
Réalisation personnelle

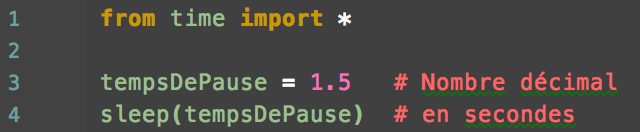
Nous nous sommes séparé le travail relativement tôt. En effet, à l’origine, je devait m’occuper de l’interface graphique, et avait d’ailleurs commencé à dessiner ce à quoi pourrait ressembler les différentes applications. Cependant, ne pouvant installer Kivy, je me suis donc concentré sur l’écriture des différents programmes de notre application.

Celle-ci requérant la gestion du temps, nous avons donc fait quelques recherches sur les bibliothèques de python permettant la gestion du temps. Nous en avons trouvé une, la bibliothèque time, intégrant des fonctions permettant de mesurer le temps.

J’ai donc commencé, à rédiger un petit programme qui a servi de base pour le chronomètre.

La fonction time.time() que nous avons utilisé, est une fonction qui permet d’assigner à une variable (entre autres) par l’écriture : variable = time.time() le nombre de secondes écoulées depuis le 1er janvier 1970 à 00 : 00 : 00 UTC (heure 0 UNIX) . En assignant plus tard à une autre variable le nombre de secondes écoulées depuis l’heure 0 UNIX, et en faisant une soustraction des deux, on peut trouver le temps écoulé entre l’attribution des deux variables :

La fonctions sleep() que nous avons également utilisé permet quand à elle de mettre en pause le programme, de l’arrêter pendant le temps indiqué en paramètre, avant de continuer. Cette fonction nous a permis pour le chronomètre de bloquer le programme pendant 1 seconde avant chaque mouvement d’aiguille.

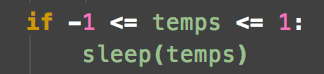


À ce stade du chronomètre, nous nous sommes rendu compte que le chronomètre n’était pas suffisamment précis, car toutes les fonctionnalités du programme prenaient un peu de temps à s’exécuter ce qui faisait perdre au programme de la précision. L’une des solutions que j’avais soulevée a été de mesurer exactement le temps que le programme mettait à s’exécuter pour ensuite soustraire ce temps à la mesure du chronomètre, mais le problème étant que chaque machine ayant une vitesse différente, il fallait donc intégrer ce calcul au programme, ce qui est relativement difficile à mettre en place. C’est à ce moment que j’ai arrêté de me concentrer sur le programme du chronomètre que Quentin et Maxime ont pris le relai pour le corriger et y intégrer une interface graphique.

J’ai donc commencé à réfléchir à la deuxième application de notre application, celle de gestion de temps/diapositive. Celle-ci a pour fonctionnalité de prévenir l’utilisateur, à un laps de temps régulier choisi par celui-ci, par une petite vibration (dans le projet original) de son téléphone. Si nous ne parvenons pas à faire vibrer le téléphone, nous avertirons celui-ci d’une autre manière, en affichant un écran blanc, par exemple. Cette application a pour origine le fait que dans une réunion, durant une présentation, le présentateur n’a pas forcément une idée claire du temps qui s’écoule réellement sans consulter de montre ou son téléphone. Le fait d’avoir une application l’avertissant du temps écoulé lui permettrait de pouvoir se rendre compte réellement du temps qui s’écoule, car la perception du temps n’est pas la même en période de stress.

J’ai donc commencé à travailler sur cette application.

Pour résumer son utilisation, on demande à l’utilisateur le nombre de période, c’est à dire le nombre de fois que le téléphone vibrera en tout, et le temps, en secondes, entre chaque période. Le programme s’arrête dès que le nombre de période est écoulé, ou que l’utilisateur décide de l’arrêter. Nous avons tout d’abord rencontré le même problème de précision que pour le chronomètre, mais ce problème a été résolu dans ce cas précis par l’utilisation d’un intervalle puis d’une fonction sleep() comme suit :



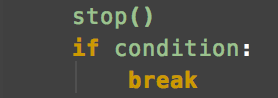
On créée délibérément une marge d’erreur, car on sait que notre résultat est imprécis, puis on met en pause le programme durant le temps qu’il nous reste pour obtenir une précision très appréciable (quasiment exacte).

Cependant, un autre problème s’est posé à nous : la fonction stop(). Il s’agit de la fonction qui casse la boucle et qui arrête le programme, en contenant un break. Cependant, un break ne peut être créé que dans une boucle, et une fonction ne peut donc pas contenir de break sans contenir de boucle, ce qui rendrait la fonction inutile, car il n’existe en python pas de break global, et s’il en existait un, nous ne l’aurions pas utilisé, car il aurait stoppé l’intégralité du programme, ce qui aurait fait crasher l’application toute entière. Il a donc fallu utiliser un autre moyen pour que l’utilisateur puisse stopper le décompte :

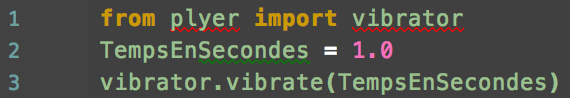
Pour résoudre ce problème, on a posé les variables booléennes condition (initialisée à False) et buttonStop (initialisée à True), et posé la fonction stop() :



Quand l’utilisateur appuie sur l’écran, la variable booléenne buttonStop va passer à False, ce qui signifie que la variable condition va passer en True. Ensuite, on écrit, entre chaque ligne de code de notre boucle :

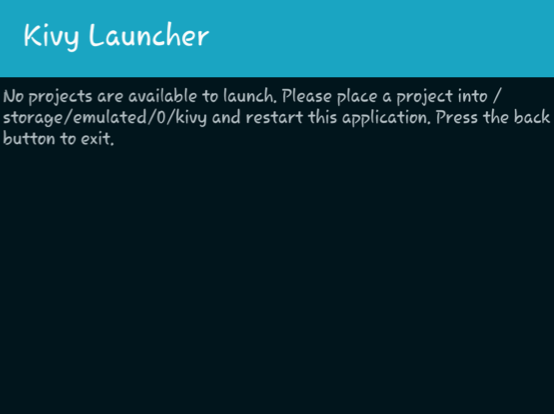


Ce qui arrête le programme. Il aurait bien sûr été plus rapide de l’écrire, sans fonction mais le fait de pouvoir lire « stop() » permet de bien se rendre compte que le bloc peut arrêter le programme. C’est plus lisible, de notre point de vue. Même intégré dans les différentes boucles du programme, la fonction va stopper l’application et permettre de la réinitialiser, car a été intégré au programme à la sortie de chaque boucle le bloc avec le fonction stop(), on sort donc de toutes les boucles.

Cependant, le problème principal rencontré sur l’écriture de ce programme est la fonction qui fait vibrer le téléphone. En effet, notre objectif étant de faire vibrer le téléphone après un temps écoulé décidé par l’utilisateur, il nous fallait intégrer au programme une fonction reconnue et par le téléphone et par le langage python (avec la bibliothèque Kivy). Après de longues recherches, nous avons trouvé la bibliothèque plyer (pour python). Cette petite bibliothèque que nous avons trouvé sur Github permet d’accéder sur le papier aux fonction d’un téléphone, sur android (nous ne savons pas si elle agit également sur IOS). Pour faire vibrer le téléphone, on utilise l’écriture :

Cependant, nous avons fait face une fois de plus à de nombreux problèmes, dont le fait que nous n’avons à l’heure actuelle pas réussi à faire passer ne serai-ce qu’une ligne de code sur un téléphone. Du coup, Internet ne répondant pas aux questions « la bibliothèque plyer est-elle intégrée au pack Kivy installé au préalable sur le téléphone ? Sinon le téléphone peut-il comprendre la bibliothèque plyer ? », il nous fallait faire des tests sur un téléphone.

Cependant, comme dit ci-dessus, nous n’avons pas réussi à faire passer l’application sur un téléphone. Nous avons cependant commencé par installer Kivy sur mon téléphone, puis, en ouvrant l’application on obtient ceci :

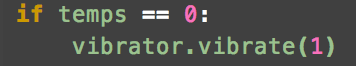


Ainsi, on connecte le téléphone à un ordinateur, et on cherche en premier lieu un fichier nommé storage, qui n’existe pas, aussi bien sur la carte mémoire (micro sd) que sur la mémoire de l’appareil. Nous créons donc tous les dossiers correspondant et plaçons le projet (de test) dans le dossier kivy, nouvellement créé. Précisons qu’aucun dossier kivy n’existait auparavant. Il ne se passe rien. Une autre solution consisterai à utiliser linux (par une machine virtuelle), mais ceci nécessite des compétences que je ne possède malheureusement pas et que je n’ai pas le temps d’acquérir. Je passe donc la main à Maxime qui s’y connaît dans le sujet. Je transmet également le programme de l’application gestion de temps/diapositive à Quentin et me met à travailler avec lui pour l’interface graphique de ce programme, qui sera bien moins poussée que pour le chronomètre, puisque nous ne comptons rien afficher, et juste placer un bouton qui fera la taille de l’écran et qui stoppe l’intégralité du programme si on touche l’écran (bouton lié au bloc d’arrêt du programme décrit précédemment). Dans le cas où on ne parvienne pas à faire vibrer le téléphone, nous remplacerons par un écran blanc ou un message affiché à l’écran la vibration.

Intégration et validation

Personnellement, je me suis plutôt occupé de l’écriture des programmes de base des applications, ainsi que des recherches sur l’installation de l’application sur android. Quentin et Maxime se sont ensuite chargés d’intégrer ces programmes a l’interface graphique. Nous n’avons malheureusement pas eu le temps de développer l’application dite ‘sport’. Cependant, nous avions pour objectif de réaliser un chronomètre et l’application « diapositive/gestion de temps », et nous pouvons considérer ces objectifs comme presque atteints (à l’huer où j’écris ces lignes, le problème de l’interface graphique du chronomètre est toujours présent, mais sans celle-ci, le chronomètre fonctionne et est précis. De plus, le programme de « gestion de temps/diapositve » est lui aussi, pour le code uniquement, parfaitement fonctionnel, après de nombreux tests.

Pour vérifier que le programme fonctionnait, ou, sinon, pour se rendre compte des erreurs glissées dans celui-ci, il a fallu faire de nombreux tests, relativement variés, en fonction des étapes du programme. En premier lieu, après avoir écrit le premier jet du programme, on a voulu vérifier si celui-ci fonctionnait (sans fonction stop() et sans vibration). Après des essais sur la précision du programme, nous nous sommes rendu compte en premier lieu qu’il ne s’arrêtait pas du tout à la fin du temps déterminé par l’utilisateur, ce qui signifiait que la variable dépassait le 0 et se décrémentait à l’infini  (nous avons pour observer le moment de vibration remplacé le bloc « vibrator.vibrate(1) » par « print(‘vibration’) »):



Nous avons donc rendu arbitrairement défini un intervalle dans lequel nous étions sûrs que la variable temps tomberait, puis nous avons utilisé un sleep() pour gagner en précision. Après plus de 50 essais montre en main pour juger de la précision du programme qui, ne l’oublions pas, importait peu si une petite précision apparaissait, nous avons observé un degré de précision quasi parfait (nous n’avons observé aucune imprécision).

il a fallu ensuite tester la fonction stop(), mais en remplaçant la variable booléenne buttonStop par une variable entière (un integer) qu’on décrémentait à chaque tour de boucle. Ainsi, on a pu observer les erreurs du programme énoncées auparavant, et donc les corriger, jusqu’à ce qu’on obtienne un résultat attendu, c’est à dire que la fonction stop() arrête le programme lorsqu’on l’a décidé. On a pu ensuite, avec Quentin, vérifier la fonction stop() par un simple clic lors de l’exécution du programme, et ça fonctionne. Nous n’avons malheureusement pas encore pu tester la vibration. Si il s’avérait impossible de faire vibrer le téléphone, nous remplacerons par l’apparition d’un image à l’écran, par exemple.

Bilan et perspectives

Les programmes créés ont pour objectif de mesurer le temps. Le chronomètre, tout d’abord, est un chronomètre à aiguilles, ce qui est différent par rapport aux chronomètres numériques actuels. De plus, l’application diapositive/gestion de temps, dans sa version telle que nous l’avons pensée à l’origine, c’est à dire avec les vibrations, permet à l’utilisateur de se rendre compte précisément du temps qui s’écoule, en réunion ou durant une présentation par exemple, où le stress modifie notre perception du temps. Pour l’interface utilisateur, on souhaite que l’application demande à celui-ci le nombre de périodes, ainsi que le temps, en seconde, par personne. Sachant que l’application se veut instinctive, il serait intéressant de rajouter une fonction qui demanderai à l’utilisateur comment il souhaite rentrer le temps, en secondes ou en minutes, puis, dans le cas des minutes, de les convertir automatiquement en secondes. Nous n’avons pas eu le temps de rajouter une telle fonction, mais si on avait eu un peu plus de temps, ce genre de fonction aurait été intégré au programme.

Également, si nous avions eu plus de temps, nous aurions développé également la troisième application de notre projet : l’application sport, qui permet à l’utilisateur de déterminer un nombre important d’activités et de les noter, comme pour une to-do-list, mais en déterminant un temps à consacrer à chacune, par exemple, l’utilisateur peut programmer « 10 minutes de corde à sauter/ 2 minutes de pause/5 minutes de squats… » à chaque temps écoulé, le programme fait sonner pendant un cour laps de temps le téléphone, lui indiquant ainsi de passer à l’activité suivante, en intégrant peut-être également un lecteur sonore de texte, un peu comme Google traduction ou acapella box.

En faisant ce projet, j’ai appris à me servir du langage python de manière plus poussée que ce que nous avions fait sur France IOI, dans le sens où j’ai fait des recherches sur certaines bibliothèques spécifiques de python, comme par exemple la bibliothèque time. Au lieu de développer des compétences plus globales ne python , j’ai plutôt obtenu des comptétences plus spécifiques, sur les bibliothèques time, plyer, et un peu kivy. Également, j’ai découvert que le travail en équipe est plus motivant. En effet, j’ai souvent eu l’habitude de travailler seul sur des projets. Quand on travaille en groupe, la motivation est plus importante qu’en solo : on ne veut pas décevoir les autres membres du groupes et donc on travaille plus, et, également, comme on abat une plus grande charge de travail ensemble, on va plus loin.