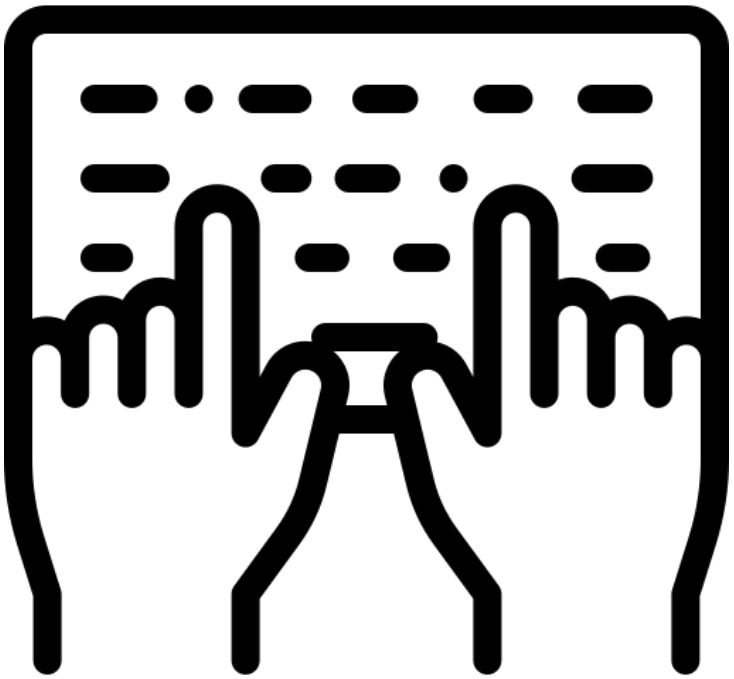
typ.

Logiciel d'entraînement à la frappe au clavier



Groupe E4
BAUS Philippe
GINEVRA Charly
HIETANEN Elias
OLENDER Cédric
VALENTIN Pierre

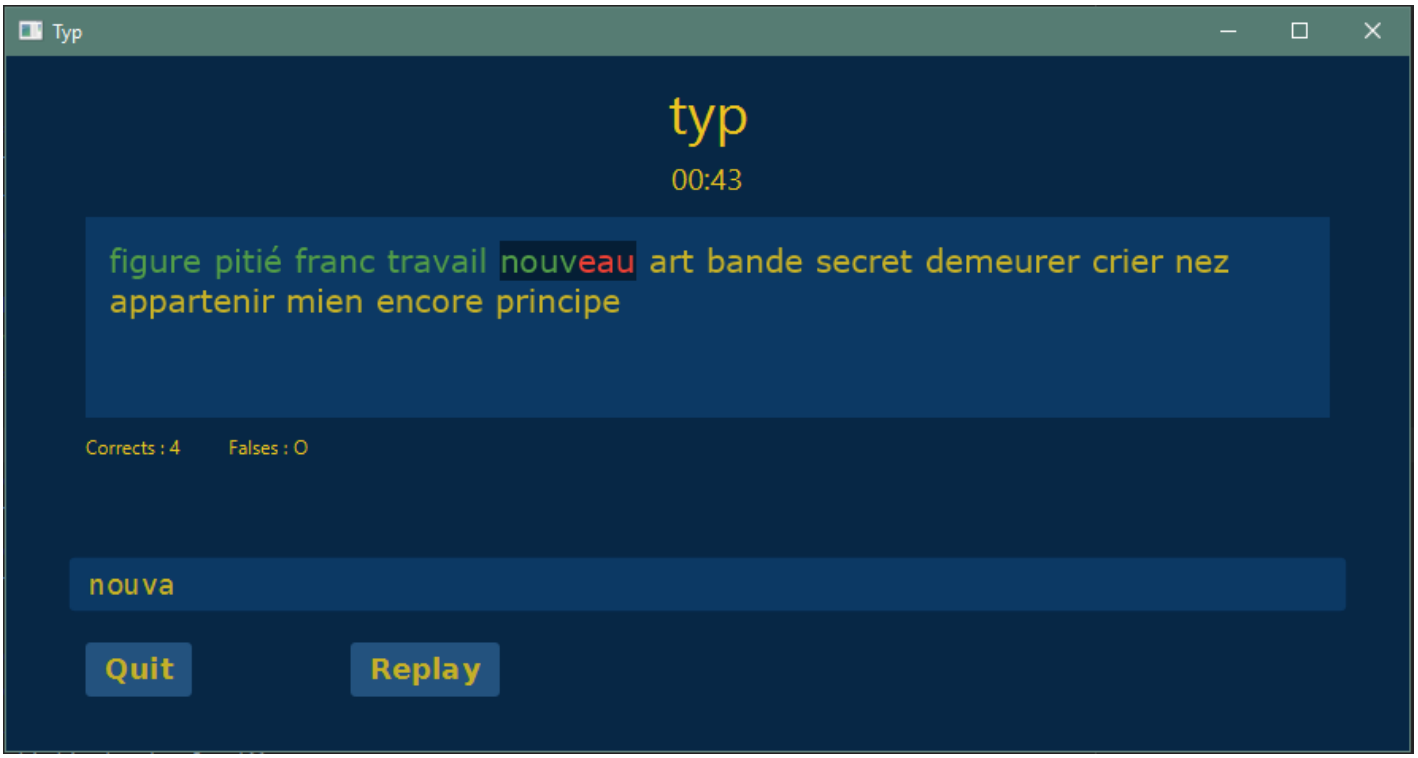
2021

Qu'est-ce que typ ?

typ est un logiciel d'entraînement à la frappe au clavier. Son but étant de proposer différents exercices à l'utilisateur, qu'il soit débutant ou avancé, afin de l'aider à écrire de manière plus confortable sur un clavier. Les axes d'amélioration sont divers, le logiciel peut aider l'utilisateur à écrire sans regarder le clavier avec ses yeux, développer une vitesse de frappe plus grande, limiter les fautes de frappe ou encore améliorer sa dextérité avec ses doigts.

typ est né de notre envie de développer un logiciel local, simple d'utilisation, ergonomique et user friendly. Notre project owner, Elias Hietanen, souhaitait développer un programme d'entraînement à la frappe au clavier qui lui convenait ergonomiquement (police grande, jeu de couleurs démarquées pour la signalisation, statistiques avancées). Notre objectif principal est que *typ* soit facile à prendre en main et qu'il propose des exercices variés et adaptés à chacun.

Notre ambition à long terme est de proposer plusieurs modes d'entraînement, de difficulté paramétrable. *typ* serait en mesure d'être modulé comme l'utilisateur le souhaite (police, taille, couleur), il proposerait un rapport de statistiques détaillé de chaque atelier (Nombre de mots justes, nombre de mots faux, Word Per Minute, etc.). Le logiciel propose plusieurs langues afin de toucher un plus grand nombre de personnes.



Outils utilisés

Trello

https://trello.com/b/R9apGsoA/typ
Dans le cadre de ce projet, il nous a été demandé de travailler avec les méthodes agiles. Afin de garder trace de notre progression de travail, de nos tâches en cours et des sprints à venir, nous avons décidé d'utiliser un tableau Trello qui permet de recenser tous les tickets de développement et les trier dans des colonnes.

Gitlab

https://gitlab.com/cedric.olender/typ
GitLab est un logiciel open source basé sur git. Git est une technologie qui permet de créer un répertoire de travail pour notre application. Chaque développeur peut travailler sur une branche qui correspond à une spécificité de l'application, pour ensuite regrouper le travail de chacun sur la branche principale et faire marcher notre application.

IntelliJ IDEA

Dans le cadre de notre projet, il nous a été imposé de développer sous le langage de programmation Java. IntelliJ est un IDE Java moderne qui propose des fonctionnalités et aides au développeur pratique pour développer plus rapidement. Nous avons décidé de choisir IntelliJ car nous préférons l'ergonomie et les fonctionnalités plus faciles à utiliser que sur l'IDE Eclipse.
SonarLint : SonarLint est une extension pour l'IDE qui permet de détecter et de résoudre les problèmes de qualité de code.

Scene Builder

Pour faciliter la construction de nos fenêtres ainsi que la mise en forme de notre user interface, nous avons utilisé l'outil Scene Builder. Cet outil nous permet de construire nos fenêtres et de les éditer à la main dans l'application, et générer le FXML de ce rendu pour l'importer dans notre projet.

User Stories et features

Feature	VM	PE	Complétion
Saisie du texte	8	4	Fait
Correction du texte mot par mot	6	5	Fait
Affichage des mots correctes et incorrectes	7	3	Fait
Génération de textes aléatoires	8	6	Fait
Relancer une partie de manière conviviale	3	2	Fait
Apparence rudimentaire	2	1	Fait
Correction du texte lettre par lettre	5	4	Fait
Affichage de la partie correcte et incorrecte d'un mot	8	8	Fait
Affichage du nombre de mots correctes et du nombre de mots incorrectes	5	1	Fait
Apparence du mode classique	5	4	Fait
Menu convivial	6	5	Fait
Plusieurs dictionnaires pour la génération de textes	4	1	Fait
Chronométrage de la partie	4	6	Fait
Génération des statistiques	4	8	Fait
Affichage des statistiques en fin de partie	6	4	Fait
Apparence du menu	5	2	Fait
Affichage du WPM (word per minutes)	4	4	Fait
EPIC : Proposer différents modes de jeux	12	15	Non Fait
Option changer la difficulté	3	7	Non Fait
Option changer de langue	5	3	Partiellement
Enregistrer et agréger les statistiques	4	5	Partiellement
Affichage convivial des statistiques	6	3	Non Fait
EPIC : Système de trophées	8	15	Non Fait
Configuration de l'apparence	5	8	Non Fait

VM : valeur métier, PE : points d'effort

Découpage de l’application en sous-systèmes

Pour réaliser *typ.* Nous avons utilisé le patron de conception Modèle Vue Contrôleur. Ainsi, nos principaux paquetages sont les paquetages Model, View et Controller. Nous avons aussi un paquetage contenant tous les tests.

Architecture des paquetage détaillée :

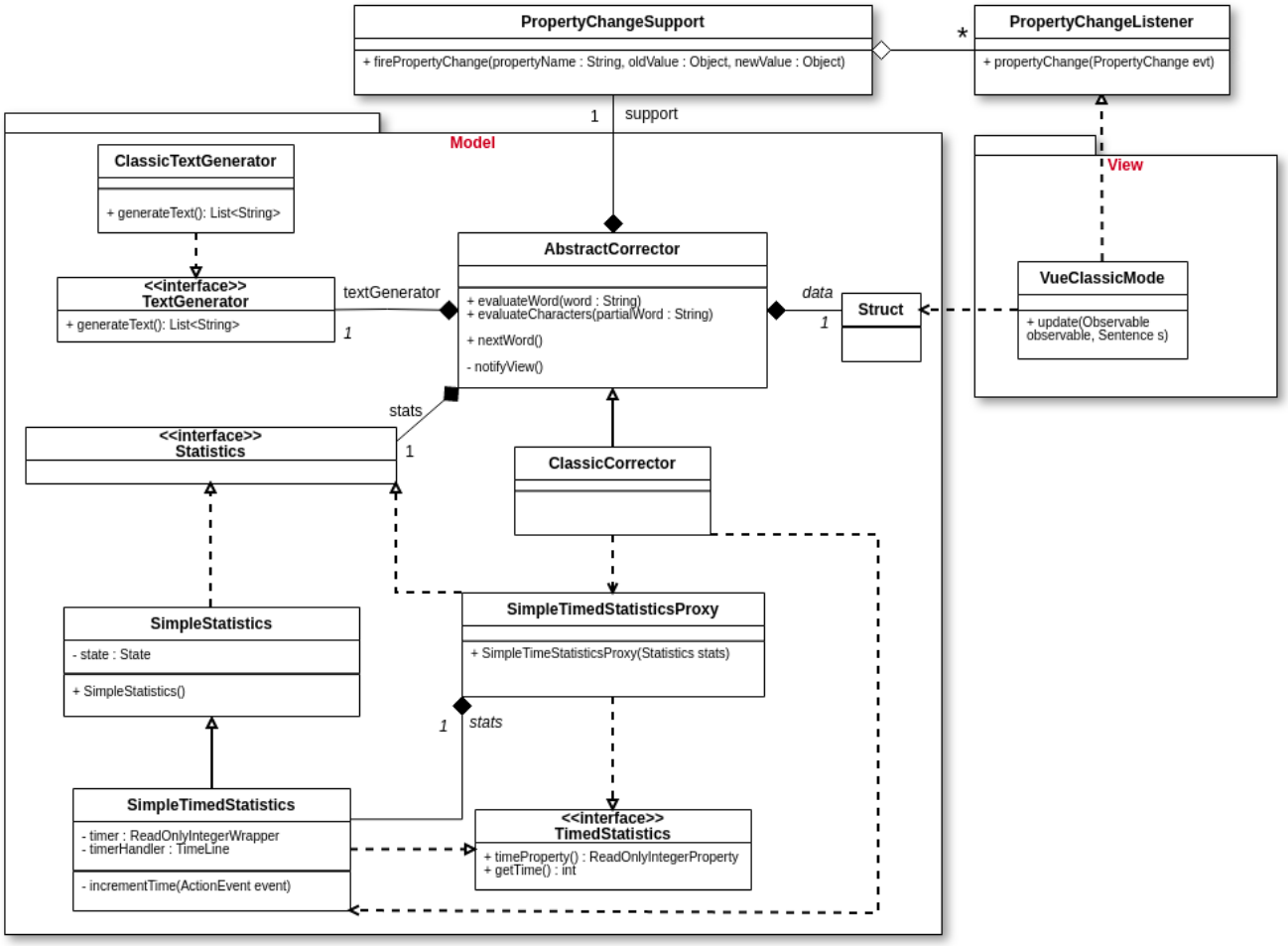
- Controller : contient les classes du controller. Le controller gère les évènements qui surviennent pendant une partie. Il indique au modèle les actions que celui-ci doit effectuer.
    - Menu : contient toutes les classes relatives au menu graphique. Ce paquetage se base sur le patron de conception Commande afin de pouvoir ajouter ou supprimer des fonctionnalités plus facilement.
- Model : contient les classes du modèle. Il y a un modèle différent par mode de jeu. Le modèle gère le texte à saisir, valide la saisie, indique le mot courant… Lorsque des modifications surviennent dans le modèle, celui-ci notifie la vue.
- View : contient toutes les classes de la vue. La vue gère l’affichage uniquement. Les éléments de la vue se mettent à jour lorsqu’ils reçoivent des notifications des éléments du modèle.
- Tests : contient toutes les classes de test

Ce découpage sera précisé dans la partie Diagramme de classe.

Fonctionnement de l’application

Lorsqu’une partie est lancée : Le modèle va notifier la vue et lui indiquer le texte à saisir et quel est le mot courant attendu. La vue va se mettre à jour, afficher le texte et surligner le mot courant attendu.
Quand l’utilisateur saisit du texte dans le champ de saisie, à chaque fois qu’un caractère est saisi, le controller va demander au modèle d’évaluer le mot. Ce dernier va se mettre à jour, et reconnaître quelle partie du mot est correcte, et quelle partie du mot est erronée. A chacune de ces modifications, le modèle va notifier ses observateurs, en l’occurrence la vue. Avec ces informations la vue va indiquer visuellement ces informations à l’utilisateur quelle est la partie du mot correcte, et quelle est la partie erronée du mot. A ce moment, l’utilisateur a toujours la possibilité de modifier le mot. **[voir annexe 3]**
Lorsque l’utilisateur appuie sur la touche espace, le controller va demander au modèle d’évaluer définitivement le mot, et de passer au mot suivant. Il va ensuite notifier la vue. Le mot sera alors intégralement vert s’il est correct et rouge s’il est incorrect. Un nouveau mot sera alors surligné. **[voir annexe 4]**
Quand la partie est terminée, une fenêtre popup s’affiche, avec les statistiques. Lorsque la fenêtre popup est fermée, une nouvelle partie se lance.

Diagrammes de classe



Note : le diagramme de classe complet est disponible à cette adresse :

Principaux choix

JavaFX vs Swing

Un de nos axes de conception majeur pour notre logiciel était l'ergonomie et le visuel. Comme dit dans l'introduction, nous souhaitons que *typ* s'accommode à tous les utilisateurs. Ainsi, nous avons décidé d'utiliser JavaFX comme technologie pour le visuel de notre application car il propose des visuels plus modernes.

Pour combler notre manque de connaissance sur l'utilisation de cette dernière, nous avons tous décidé au début du développement du projet de faire une "veille technologique".
Une "veille technologique" consiste à se renseigner sur une technologie, comprendre son utilisation et apprendre les bases pour l'utiliser convenablement.

Quand bien même nous avons décidé d'utiliser JavaFX au lieu de Swing, nous avons tout de même décidé de développer sous une architecture MVC similaire à celle vue en cours cette année avec Mme. Hurault.

Basculement vers fxml

Dans le but d'accélérer la conception et le développement des éléments graphiques. Nous avons décidé d'utiliser la technologie *fxml*. Cette dernière permet de décrire un composant graphique au sein d'un fichier ayant une syntaxe similaire à *xml*. Il est, par la suite, chargé au sein de notre programme et utilisé de manière similaire à un composant normal.

Un outil, créé par Oracle et intégrable à Intellij, appelé le Scene Builder (voir annexe 3), est disponible pour aider à la création de ces fichiers de manière graphique.

Cependant, comme nous n'avions pas fait ce choix dès le début. Par conséquent, lors du sprint 2, il a fallu effectuer des changements dans la structure de notre application pour supporter fxml.

Patrons de conception

Pour le développement de *typ.* nous avons utilisé trois patrons de conception

- **MVC** : L'architecture *Modèle/Vue/Contrôleur* (MVC) est une façon d'organiser une interface graphique d'un programme. Elle consiste à distinguer trois entités distinctes qui sont, le *modèle*, la *vue* et le *contrôleur* ayant chacun un rôle précis dans l'interface. Il est utilisé sur l'ensemble du projet.
- **Command** : Patron qui permet de séparer complètement le code initiateur de l'action, du code de l'action elle-même. Il est utilisé pour le menu de l'application.
- **Proxy** : Un proxy est une classe se substituant à une autre classe. Il est utilisé principalement pour contrôler l'accès aux méthodes de la classe substituée. Dans notre cas, cela permet à la vue de connaître les statistiques mais pas de les incrémenter.

Fonctionnement du MVC

Dans un premier temps, nous avions réalisé notre MVC exactement comme appris en cours. Notre vue implémentait l'interface *Observer* , notre modèle était un *Observable*. Le controller indiquait au modèle quelles actions il devait effectuer, et lorsque ce dernier se mettait à jour, il notifiait la vue.
Cependant, nous avons constaté que le duo *Observer / Observable* n'était pas assez flexible à notre goût. Nous avons gardé la même organisation mais avons décidé d'utiliser des *PropertyChangeListener* et des *Property* de javafx. Comme cela, nous avons gardé un patron Observer, mais à l'échelle des attributs et non plus de la classe entière.

Choix visuels

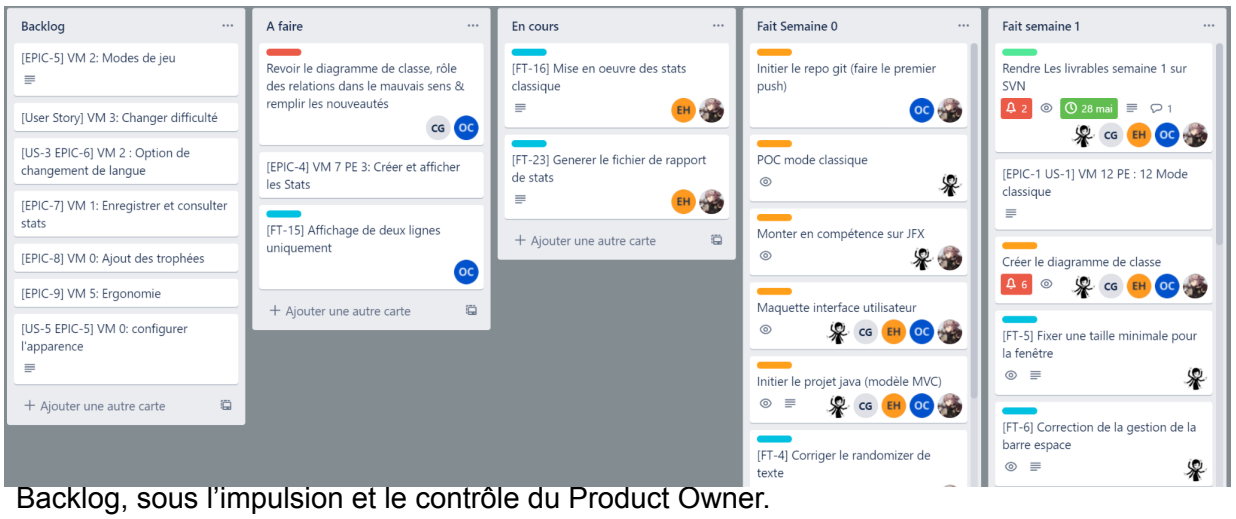
Les choix visuels sont des choix clés pour *typ.* Au départ, nous avons décidé de suivre la charte visuelle faite par notre project owner. Elle convient aux besoins primaires de l'application dans un premier temps. Enfin, l'objectif à long terme de l'application est de proposer un visuel modulaire pour convenir aux besoins de chaque utilisateur et couvrir des situations comme le handicap par exemple (daltonisme, dyslexie, dysorthographie etc.).

Organisation de l'équipe et mise en place des méthodes agiles

Organisation de l'équipe

Elias HIETANEN : Développeur, Project Owner, Admin Trello
Pierre VALENTIN : Développeur, Admin Git
Cédric OLENDER : Développeur, Admin Git
Philippe BAUS : Développeur, Responsable Communication
Charly GINEVRA: Développeur, Scrum Master

Kanban (Trello)



Backlog, sous l'impulsion et le contrôle du Product Owner.

A Faire : correspond au Backlog de sprint. Lors du sprint, les US et EPIC sélectionnés sont déplacés ici. A partir de là, les US vont être décomposées en tickets de développement.
En cours : les tickets en cours de traitement dans le sprint.
Fait semaine X : les tickets complétés la semaine X. Une US est mise dans cette colonne quand tous ses tickets de développement sont terminés. Une EPIC est mise dans cette colonne quand toutes les US relatives à l'EPIC sont terminées.
(Lien du Trello pour une meilleur lisibilité : https://trello.com/b/R9apGsoA/typ)

Organisation en sprints

Le développement de *typ.* s'est déroulé en 4 itérations d'une semaine. Chaque itération (sprint) était découpé de la manière suivante :

- **Sprint planning :** réalisé en fin de semaine, le sprint-planning consiste à organiser le sprint de la semaine suivante. Voici comment nous procédions : Dans un premier temps, nous alimentions le backlog général avec les nouvelles idées et fonctionnalités dont nous avions eu l'idée. Ensuite, nous rebasculions les tâches de la colonne "en cours" directement vers le backlog de sprint ("A faire") sur Trello. Ensuite, nous votions pour évaluer les points d'efforts de tous les éléments présents dans le backlog général. Enfin, grâce aux points d'effort et à la valeur métier, nous déterminions les tâches "à faire" pour le sprint suivant. Nous essayons de nous assigner des US stories de manière à voir une quantité de points d'efforts estimés égale ou très légèrement supérieure (<= +2) à la quantité de points d'efforts réalisée lors du sprint précédent

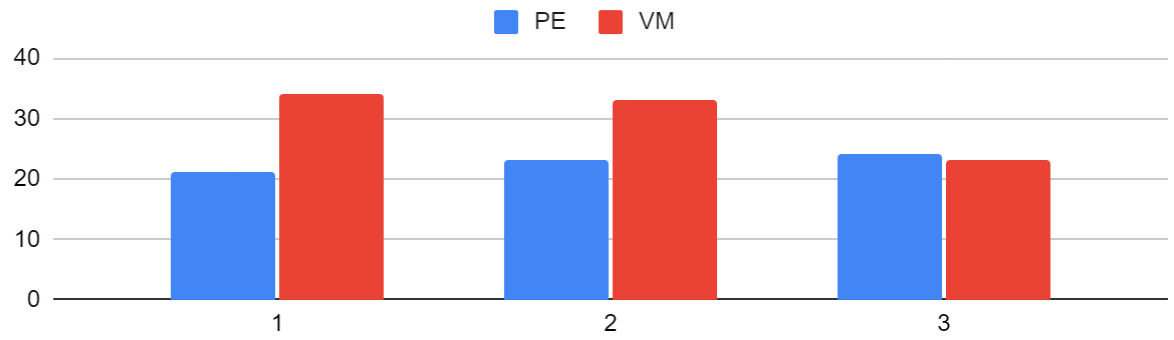
- **Daily scrum :** tous les jours, nous nous sommes réunis lors de courtes réunions appelées "Daily Scrum". Lors de ces réunions, chaque membre de l'équipe présentait ce sur quoi il avait travaillé la veille, les difficultés rencontrées et le travail envisagé pour la journée.

- **Sprint sanity :** En milieu de semaine, généralement le mercredi ou le jeudi, nous réalisions un sanity check pour le sprint en cours. Cela consistait en un vote pour savoir si nous étions confiant quant à la réalisation des tâches prévues pour le sprint.

- **Sprint Review :** Tous les vendredis soir, nous nous retrouvions afin de faire le bilan du sprint écoulé. Dans un premier temps, nous regardions le kanban pour évaluer notre vélocité. Cela serait utile pour préparer le sprint suivant. Ensuite, nous regardions si nous avions réussi à atteindre nos objectifs pour la semaine. Enfin, nous procédions simplement à un tour de table. Chacun son tour, nous exprimions brièvement les points que nous avions appréciés et les points à améliorer. Le cas échéant, nous prenions des mesures palliatives. Par exemple, suite à une Sprint Review, nous avons instauré un protocole GIT plus strict à respecter.

Détails des sprints

n°	User story	**VM**	**PE**
1	Saisie du texte	8	4
	Correction du texte mot par mot	6	5
	Affichage des mots correctes et incorrectes	7	3
	Génération de textes aléatoires	8	6
	Relancer une partie de manière conviviale	3	2
	Apparence rudimentaire	2	1
	Bilan	**34**	**21**
2	Correction du texte lettre par lettre	5	4
	Affichage de la partie correcte et incorrecte d'un mot	8	8
	Affichage du nombre de mots correctes et du nombre de mots incorrectes	5	1
	Apparence du mode classique	5	4
	Menu convivial	6	5
	Plusieurs dictionnaires pour la génération de textes	4	1
	Bilan	**33**	**23**
3	Chronométrage de la partie	4	6
	Génération des statistiques	4	8
	Affichage des statistiques en fin de partie	6	4
	Apparence du menu	5	2
	Affichage du WPM (word per minutes)	4	4
	Bilan	**23**	**24**



PE et VM en fonction du sprint

User Stories et Epic toujours présentes dans le backlog :

EPIC : Proposer différents modes de jeux	12	15	Non Fait
Option changer la difficulté	3	7	Non Fait
Option changer de langue	5	3	Partiellement
Enregistrer et agréger les statistiques	4	5	Partiellement
Affichage convivial des statistiques	6	3	Non Fait
EPIC : Système de trophées	8	15	Non Fait
Configuration de l'apparence	5	8	Non Fait

Conclusion sur la gestion Agile

Lors de ce projet, nous avons essayé d'appliquer au mieux les méthodes agiles enseignées en cours. Notre expérience professionnelle en équipe agile acquise lors des alternances nous a beaucoup aidé à adopter rapidement les bonnes pratiques. L'impératif de produire des livrables chaque semaine, et la planification de sprint en amont nous a poussés à nous concentrer sur les fonctionnalités les plus essentielles. La sprint review et le sanity check nous ont permis de mettre en place rapidement et de manière dynamique des solutions adaptées à nos problèmes. Par exemple, suite à un problème de gestion git, nous avons rédigé et mis en œuvre un protocole pour éviter que le problème ne se reproduise. Enfin, les daily scrum ont permis l'échange de savoir, une meilleure répartition du travail et une vrai entraide.

Bilan

Difficultés rencontrées et solutions apportées

- Commencer le projet : le début du projet a été difficile. Nous ne savions pas par où commencer. Nous nous sommes inspirés de notre expérience en entreprise, et avons réalisé une POC. Cela nous a permis de poser les premières bases de notre application et de mieux cerner les premières problématiques.
- Git : Lors du sprint 2, nous avons eu un problème avec GIT. En effet, nous avions merge dans la branche principale des modification qui posaient problème. Pour résoudre cela, nous avons dans un premier temps dû annuler les commits un à un, puis refaire les merge proprement. Cela nous a pris beaucoup de temps, et pour que cela ne se reproduise plus, nous avons établi un protocole d'utilisation de GIT plus strict.

Compétences acquises

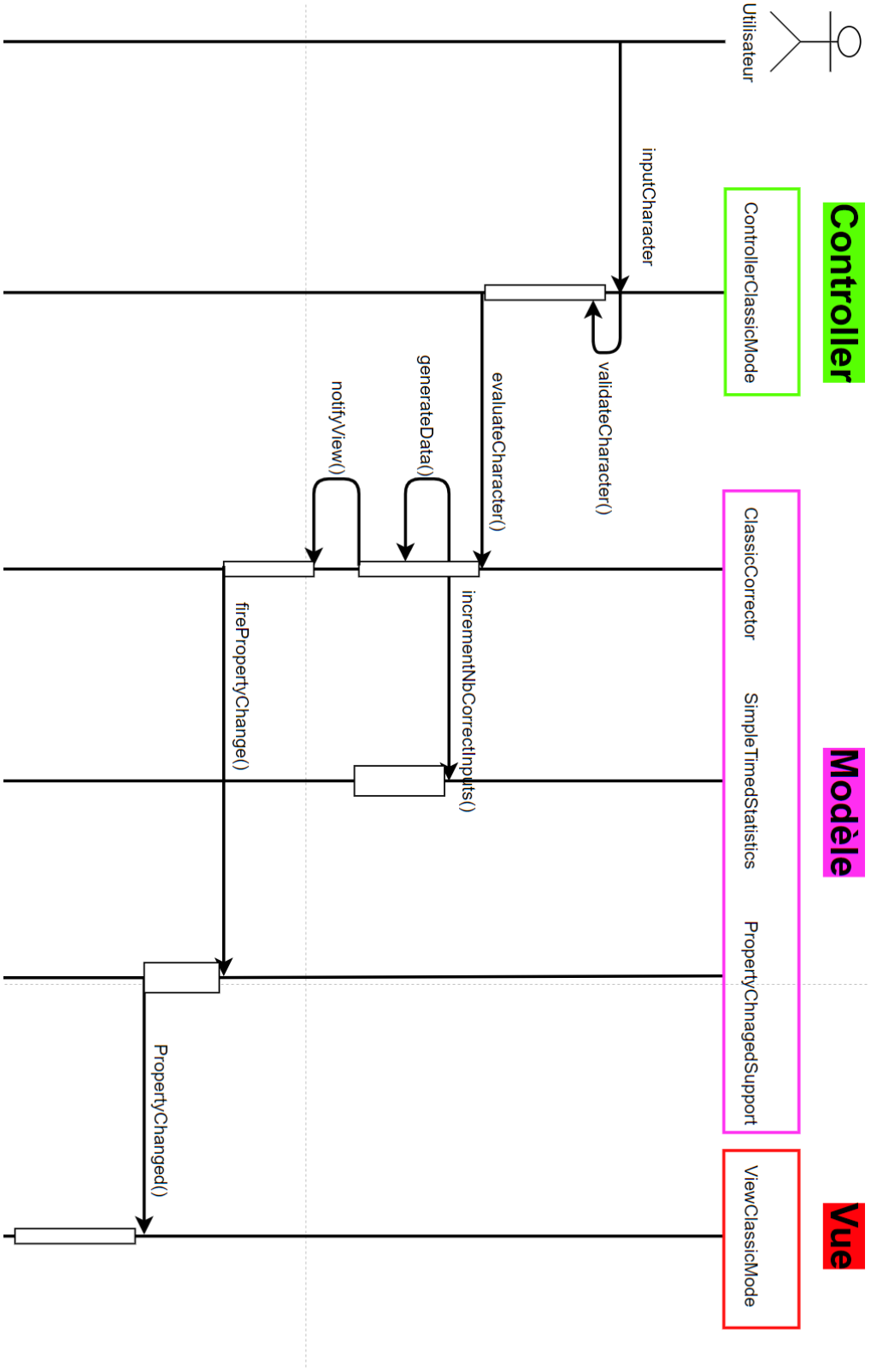
- Gestion de projet agile
- Git (stash, checkout, merge, merge requests)
- Communication et répartition des tâches
- Extreme Programming
- Application de patrons de conception
- JavaFX, Maven, FXML
- Création de tickets

Conclusion :

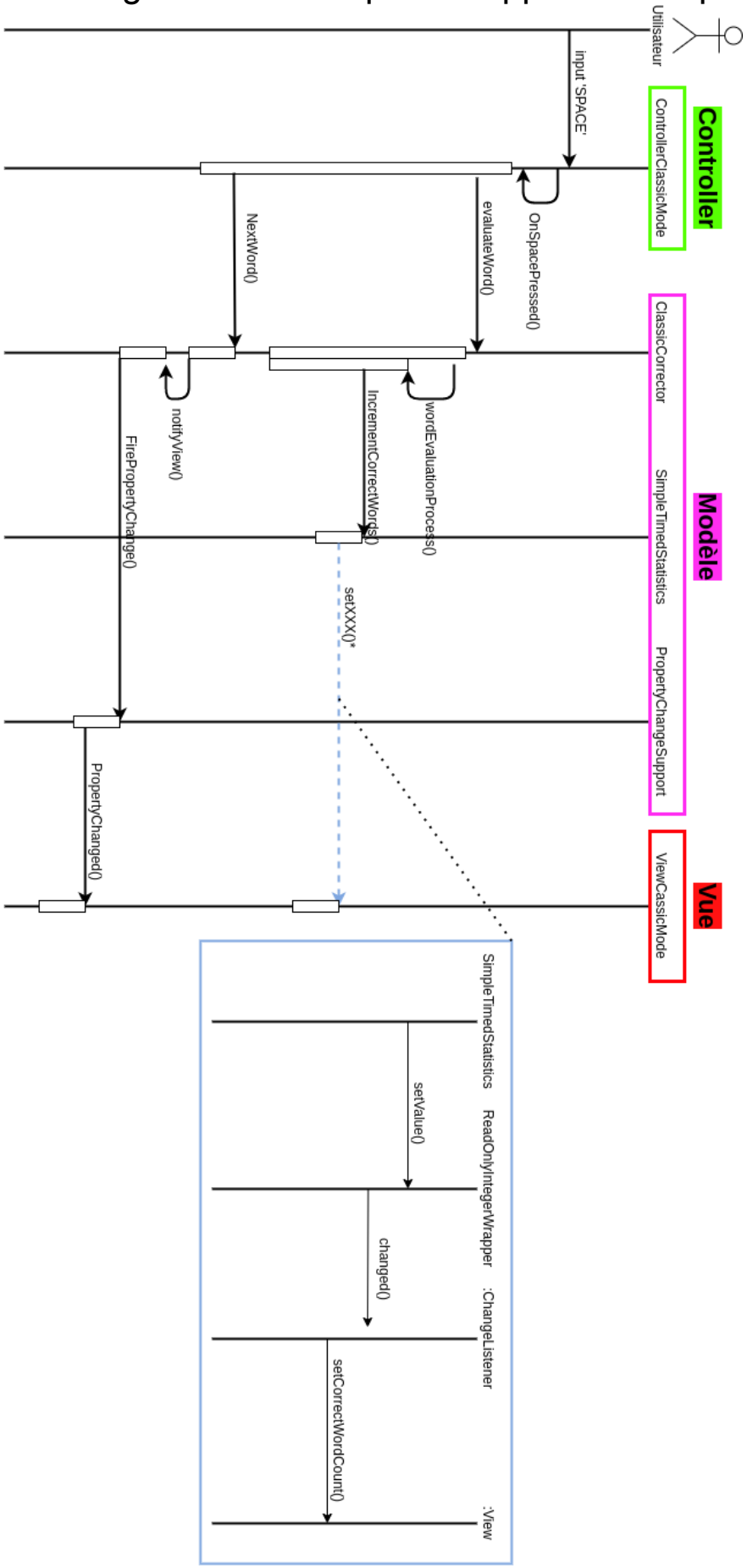
Pour conclure, *typ* est un logiciel d'entraînement à la frappe au clavier développé pour une grande diversité d'utilisateurs. Notre objectif est de toucher un grand nombre de personnes, qu'elles soient débutantes, avancées, qu'elles parlent différentes langues, ou qu'elles soient handicapées. Développé de manière modulaire, notre application nous permet d'ajouter simplement des nouveaux modes de jeu.
Enfin, nous prévoyons d'ajouter des paramètres de personnalisation visuelle de l'interface, de consulter les statistiques sous forme de graph ainsi que la possibilité de sauvegarder les statistiques afin de gérer plusieurs utilisateurs sur la même machine.

Annexes

Annexe 1 : Diagramme de séquence saisie d'un caractère



Annexe 2 - Diagramme de séquence appuie sur espace



Annexe 3: Scene builder