SAE 201 Développement d'une application

Etape 3

Rédigé par Clément LEFÈVRE, Thomas GORISSE,
Antoine FADDA-RODRIGUEZ

Semestre 2

Rapport S201

Enseignant : Léo DONATI
Département informatique
Année Universitaire : 2021-2022

Composition de l'équipe :

Clément LEFÈVRE, Thomas GORISSE, Antoine FADDA-RODRIGUEZ

Lien vers le GitHub :

[Lien vers le GitHub de l'équipe]

Description des choix algorithmiques, des difficultés rencontrées

Choix algorithmiques

- Sauvegarde des informations sous format JSON en local. Le processus consiste à récupérer toutes les informations des dresseurs à la fin d'un combat, de manière à créer une liste JSON des dresseurs ayant joué. A la nouvelle exécution du programme, les fichiers JSON sont lus de manière à récupérer les informations des dresseurs existants déjà dans les fichiers de sauvegarde. Le fait de les sauvegarder en JSON nous permet d'y accéder très simplement et rapidement, en plus d'accroître la lisibilité de notre code.
- Nous avons choisi de ne pas créer de classe Echange pour implémenter IEchange car cette interface n'implémente que peu de méthodes et aucun attribut n'est nécessaire. Nous l'avons donc implémenté dans Dresseur car c'est bien un dresseur qui change de pokémon lors d'un combat.
- expliquer la gueule de la classe combat et ses méthodes bizarres
- Nous avons choisi de faire de notre classe Dresseur une classe abstraite car aucun objet Dresseur n'est nécessaire puisque IARandom et Joueur sont, eux, des dresseurs dont ont doit créer des instances. Ainsi ces 2 classes définissent les méthodes de dresseurs qui nécessitent des inputs de l'utilisateur ou aléatoires.
- Nous avons choisi de gérer tous les inputs de l'utilisateur dans la classe InputViaScanner car le code est bien plus lisible comme cela. De plus, de nombreux bugs présents à l'utilisation des scanners, que nous n'avions pas compris, ont été réglés depuis la création et utilisation de cette classe.

Difficultées rencontrées

À travers la réalisation de cette SAÉ, nous avons rencontré plusieurs problèmes.

- Lorsque nous avons commencé à tester les combats, nous n'avions pas complètement compris ce qui devait être mis à jour lors d'un gain de niveau. Mais nous nous sommes vite rendu compte que le fait qu'un pokémon qui récupère tous ses PV lorsqu'il gagne un niveau n'était pas normal, nous ne mettons donc plus à jour le PV lors d'un gain de niveau mais simplement ses PVmax.
- Ensuite, en tentant d'échanger de pokémon pendant un combat nous pouvions envoyer un pokémon KO au combat mais aussi le pokémon qui y était déjà il a donc fallu ajouter des conditions sur ce choix
- Un des problèmes les plus importants auxquels nous avons été confrontés concerne les Scanner. C'est la classe que nous utilisons pour lire les entrées de l'utilisateur. Le

problème est que ce système nous laisse écrire dans la console même quand l'application n'a pas besoin de récupérer de valeur. Par contre, si on entre une valeur avant que le système en aie besoin, il lira la valeur précédemment entrée lorsqu'il aura de nouveau besoin d'une valeur. Il est donc important de ne pas entrer plus de valeur que nécessaire car nous n'avons pas trouvé comment régler ce problème.

- Enfin le problème majeur résidait dans les collections de capacité que chacune des espèces peuvent apprendre. De part la simplification de la liste des capacités de la première génération qui retire les capacité les plus complexe à gérer, certain pokémon se retrouve très peu de capacité voire aucune pour certain d'entre eux durant leur premiers niveaux. Nous avons donc dû ajouter l'utilisation de la capacité lutte lorsqu'un pokémon n'a pas de capacité qu'il peut utiliser.

Description des suites de tests

Avant chaque test on exécute un @BeforeEach nous permettant d'instancier à chaque test des nouveaux dresseurs avec une nouvelle équipe et des nouvelles statistiques.
Dans celui-ci se créer 2 IA Random , un pokeTest nous permettant de tester plusieurs actions sur celui-ci lors d'un combat et un combat C1.

-testEchangePokemon() a pour objectif de vérifier qu'un pokémon a bien été remplacé sur le terrain. Pour cela, nous testons d'abord que le pokémon sur le terrain n'est pas celui avec lequel on souhaite le remplacer. Ensuite, on utilise le pokeTest pour l'insérer dans l'équipe de l'IA et pour remplacer le pokémon en jeu. Puis grâce à un assertEquals on test si le pokémon sur le terrain est bien pokeTest
-testKO() a pour objectif de tester si une équipe peut encore se battre, autrement dit si tous les pokémons d'une équipe sont KO. Pour cela on les met tous KO un par un et on test si la méthode pouvoirSeBattre() grâce à un assertTrue
-testSoigneRanch() a pour objectif de tester la méthode soigneRanch(). Pour cela on met KO tous les pokémons d'une équipe puis on les soigne et on vérifie qu'ils ont tous été bien soignés grâce à un assertTrue().
-testUtilise() a pour objectif de tester la méthode utilise() qui doit faire utiliser une capacité. Pour cela, on compare les PP d'une capacité avant et après l'utilisation.
-testnbPokemon() qui compte le nombre de pokémon en vie d'une équipe. Pour cela, on met un pokémon KO et on vérifie que le nombre de pokémon d'une équipe à bien diminué.
-testTermine() qui vérifie qu'une équipe est bien est bien soignée après un combat pour cela, on met des pokémons KO et on termine le combat. On vérifie ensuite qu'ils ont bien tous été soignés.
-testCommence() On vérifie que la méthode commence bien un combat et donc que le nombre de tour est bien égal à un.
-testWinner() Il fallait tester si la méthode retournait bien un vainqueur. Néanmoins, ce sont deux IARandom qui combattent. Ainsi, les chances de victoires sont aléatoires. On vérifie donc qu'une des deux IA à bien gagné le combat.

Organisation et utilisation de votre dépôt Git

Organisation

Notre organisation réelle a évolué depuis la rédaction de notre CONTRIBUTING.md rédigé pour le rendu 2. En effet, avant même la date butoir du 2eme rendu, Antoine avait commencé la conception de la classe Combat. Alors lors du début des deux semaines de travail sur pour ce 3eme rendu, nous avons décidé que c'est lui s'occuperait de faire la conception et la programmation de cette classe ainsi que de la classe Campagne. Clément et Thomas devraient, pendant ce temps, faire la conception du système de connexion et inscription. Puis Clément ferait la conception du système de sauvegarde ainsi que sa programmation en plus de celle du système de connexion et inscription. Thomas serait alors en train de rédiger les premières parties de ce rapport, pour ensuite passer sur la batterie de test dès que Clément ou Antoine auraient fini une partie de leur code.

Nos choix sont justifiés par les goûts et compétences de chacun. Clément s'était occupé pour le rendu 2 de la lecture des JSON et même de leur écriture, il était donc naturel qu'il s'occupe du système de sauvegarde. Thomas avait lui fait tous les tests du rendu 2, sachant bien comment faire et quoi tester, il s'est donc vu charger de faire de nouveau les tests pour ce rendu. Antoine était optimiste à l'idée de travailler sur la classe Combat et ses méthodes, il s'est donc occupé rapidement de cette partie.

Stratégie d'utilisation de votre dépôt Git

Notre stratégie d'utilisation du Git est la même que dans notre CONTRIBUTING.md. Chacun a sa branche et travaille dessus. Des fusions sont faites quand quelqu'un a besoin du travail d'un autre. Puis quand une version fonctionnelle et sans bug est disponible, nous la poussons sur le main.