# Rapport d’activité

Dans notre cas, nous avons implémenter un jeu qui un peu inspiré de l’univers pokémon. L’idée c’est qu’au début du jeu, le joueur choisit les caractéristiques de son personnage (taille et genre) puis obtient un animal qu’il utilisera pour combattre jusqu’à la fin du jeu.

Pour mettre en place le jeu, nous avons utilisé la structure de nœuds décrite dans le sujet. Nous avons fait le choix, pour mieux avoir le contrôle sur l’histoire, de considérer que la trame ne se suit que sur 3 points : Choix de l’évènement suivant fait par le joueur, Choix aléatoire de l’évènement suivant et fin de jeu. Ainsi pour cela nous avons implémenter la classe InnerNode comme étant abstraite. Ainsi toute l’histoire se fera à travers cette classe mais elle ne peut correspondre que dans le cas de DecisionNode ou ChanceNode.

Pour aussi mieux contrôler notre histoire nous avons mis en place une méthode d’ajout de liste prenant en paramètre une arraylist, tout en ayant un mécanisme de validation et d'ajout de nœuds qui garantit que les données ajoutées respectent une contrainte de taille. Nous avons aussi mis une gestion d’exception dans la classe InnerNode (sur la méthode ajouter_noeud) au cas où si la taille de la liste des nœuds n'est pas égale à la constante de nombre de nœud(4 dans notre cas).

Dans notre jeu, nous avons mis en place une factory qui fournit aléatoirement un animal au personnage (il s’agit d’un choix personnel car on aurait pu modifier la méthode creerAnimalAleatoire de la classe Animal et laisser le personnage choisir lui-même son animal). Pour le système de combat, nous avons attribuer à chaque animal un élément naturel (terre, feu, air, eau) et nous avons ajouté un système de bonus d’un animal à un autre selon son élément naturel (par exemple le feu > eau > air > terre). Ce système est mis en place dans les classes de chaque Animal au niveau de la méthode Attaque (étant donné que chaque animal a sa caractéristique, nous avons créé une méthode Attaque abstraite dans la classe animal) puis nous l’avons override à chaque animal.

Pour l’initialisation du jeu, nous avons mis en place une interface IHM qui permettra donc au joueur d’interagir avec la machine pour apporter ce côté immersif. Dans la classe IHM_console, nous avons créé les méthodes permettant ainsi au joueur de faire le personnage (taille, genre) et le choix de son compagnon (qui va retourner aléatoirement donc un Animal) et nous y avons aussi intégrer les différents tout le cadre sonore à l’occurrence les bruits des différents animaux.

La classe Game représente le cœur logique de notre jeu. On y retrouve diverses composantes importées de plusieurs packages comme Representation, Univers, et ihm. On en mis en place cette classe dans le but de gérer l'initialisation du joueur, la configuration initiale du jeu, la gestion des nœuds de décision et de chance, et le déroulement de la partie. Avec l’implémentation de la méthode lancement, le jeu évolue au fur et à mesure que les nœuds sont parcourus et les actions associées sont exécutées. Ainsi, cette classe permet une certaine modularité et flexibilité grâce à l'utilisation de différentes classes de nœuds et de l'interface utilisateur, facilitant ainsi l'ajout ou la modification des composantes du jeu.

Nous avons tenté au maximum de faciliter les éventuelles évolutions. Par exemple via notre interface IHM nous facilitons la bascule vers un autre type d’ihm comme une ihm graphique. Nous pouvons facilement ajouter plus de nœuds possible en changeant simplement une constante. Nous avons aujourd’hui que 4 animaux mais nous pouvons facilement en ajouter d’autres. De même Nous avons une interface action qui est contenu et lancée dans chaque nœud qui est un combat aujourd’hui. Mais l’on peut afacilement ajouter d’autres types d’actions tel que des dialogues, des échanges de pokemon, des mini jeux. Il suffit juste de créer une nouvelle classe qui implémente l’interface Action.