

Projet Logiciel Transversal

Hélène HU – Adrien SERMET

Table des matières

1 Objectif.....	3
1.1 Présentation générale.....	3
1.2 Règles du jeu.....	3
1.3 Conception Logiciel.....	3
2 Description et conception des états.....	4
2.1 Description des états.....	4
2.2 Conception logiciel.....	4
2.3 Conception logiciel : extension pour le rendu.....	4
2.4 Conception logiciel : extension pour le moteur de jeu.....	4
2.5 Ressources.....	4
3 Rendu : Stratégie et Conception.....	6
3.1 Stratégie de rendu d'un état.....	6
3.2 Conception logiciel.....	6
3.3 Conception logiciel : extension pour les animations.....	6
3.4 Ressources.....	6
3.5 Exemple de rendu.....	6
4 Règles de changement d'états et moteur de jeu.....	8
4.1 Horloge globale.....	8
4.2 Changements extérieurs.....	8
4.3 Changements autonomes.....	8
4.4 Conception logiciel.....	8
4.5 Conception logiciel : extension pour l'IA.....	8
4.6 Conception logiciel : extension pour la parallélisation.....	8
5 Intelligence Artificielle.....	10
5.1 Stratégies.....	10
5.1.1 Intelligence minimale.....	10
5.1.2 Intelligence basée sur des heuristiques.....	10
5.1.3 Intelligence basée sur les arbres de recherche.....	10
5.2 Conception logiciel.....	10
5.3 Conception logiciel : extension pour l'IA composée.....	10
5.4 Conception logiciel : extension pour IA avancée.....	10
5.5 Conception logiciel : extension pour la parallélisation.....	10
6 Modularisation.....	11
6.1 Organisation des modules.....	11
6.1.1 Répartition sur différents threads.....	11
6.1.2 Répartition sur différentes machines.....	11
6.2 Conception logiciel.....	11
6.3 Conception logiciel : extension réseau.....	11
6.4 Conception logiciel : client Android.....	11

1 Objectif

1.1 Présentation générale

On se base sur le combat tour par tour du jeu Pokémon.

Deux Pokémon, A et B, s'affronte dans une arène. On a le choix de l'arène qui modifie le climat. On choisit une équipe de 6 Pokémon. Le choix des Pokémon se fait par une interface semblable à l'image ci-dessous :

The screenshot shows a Pokémon team management interface. At the top, a team of six Pokémon is displayed: Haxorus, Cradily, Serperior, Jellicent, Blissey, and Cresselia. Below this, the details for Haxorus are shown. The nickname is Haxorus. The details section includes Level (100), Gender (—), Happiness (255), and Shiny (No). The moves listed are Iron Tail, Earthquake, Dragon Dance, and Outrage. The stats section shows HP, Atk (252+), Def, SpA, SpD (36), and Spe (220). The item is Lum Berry and the ability is Mold Breaker. Below the details, the EVs section shows a suggested spread: Fast Physical Sweeper: 220 Spe / 252 Atk / 36 SpD / (+Atk, -SpA). The EVs are distributed as follows: HP (0), Attack (252+), Defense (0), Sp. Atk (0), Sp. Def (36), and Speed (220). The IVs are all 31. The nature is Adamant (+Atk, -SpA). The remaining EVs are 0.

Base	EVs	IVs
HP 76	0	31
Attack 147	252+	31
Defense 90	0	31
Sp. Atk. 60	0	31
Sp. Def. 70	36	31
Speed 97	220	31

Remaining: 0

Nature: Adamant (+Atk, -SpA)

Ressources

2 Description et conception des états

L'objectif de cette section est une description très fine des états dans le projet. Plusieurs niveaux de descriptions sont attendus. Le premier doit être général, afin que le lecteur puisse comprendre les éléments et principes en jeux. Le niveau suivant est celui de la conception logiciel. Pour ce faire, on présente à la fois un diagramme des classes, ainsi qu'un commentaire détaillé de ce diagramme. Indiquer l'utilisation de patron de conception sera très appréciée. Notez bien que les règles de changement d'état ne sont pas attendues dans cette section, même s'il n'est pas interdit d'illustrer de temps à autre des états par leur possibles changements.

2.1 Description des états

Un état de jeu est formé par les deux équipes de 6 Pokémon et le terrain. Le joueur choisit les actions de la première équipe, l'autre est contrôlé par une Intelligence Artificielle (ou un autre joueur).

2.1.1 État du terrain

Le terrain est choisi par le joueur avant le début du combat. Chaque terrain (Temps et/ou Champs) a un nom et des effets. Les champs n'affectent que les Pokémon au sol. Le joueur peut choisir de ne pas avoir de terrain.

Nom	Types [+ augmente les dégâts - baisse les dégâts]	Effets spéciaux
TEMPS		
Soleil	+Feu, +Plante / -Eau	Lance Soleil en 1 tour
Pluie	+Eau / -Feu	Fatal Foudre avec précision de 100%
Neige	+Glace	Blizzard avec précision de 100% -1/16 PV entre chaque tour (sauf Glace)
Sable	+Roche, +Sol	-1/16 PV entre chaque tour (sauf Sol/Roche/Acier)
Brouillard	+Normal	Baisse la précision
Vent	+Dragon, +Vol / -Insecte	Vitesse doublé pour les Pokémon en l'air Vent Violent avec précision de 100%
CHAMPS		
Herbu	+Plante, +Insecte / -Sol	Récupère 1/16 de PV entre chaque tour
Électrique	+Électrique	Sommeil impossible
Psychique	+Psy	État spécial impossible
Brumeux	+Fée / -Dragon	Attaque de priorité impossible

2.1.2 État des joueurs

Élément Pokémon

Un joueur contrôle un Pokémon. Il choisit l'attaque qu'il utilise et l'objet qu'il tient. Chaque Pokémon possède

- un nom
- un id
- 4 attaques
- un ou deux types parmi Électrique, Psy, Poison, Spectre, Eau, Glace, Sol, Roche, Combat, Acier, Ténèbres, Feu, Plante, Insecte, Normal, Vol, Dragon, Fée
- un talent : ils peuvent modifier les statistiques du Pokémon, infliger un état spécial, placer un terrain, modifier le type du Pokémon ou de son attaque, soigner, donner des immunités ou infliger des dégâts.
- un objet
- des statistiques : PV, attaque, défense, attaque spéciale, défense spéciale, vitesse. Ces statistiques permettent de calculer les dégâts infligés et reçus par une attaque
- un état spécial : Aucun, Paralysie, Brûlure, Sommeil, Gel, Poison, Confusion

Nom	Descriptif des états spéciaux
Paralysie	Divise la vitesse par 2 A chaque tour le Pokémon a une probabilité de 25% de ne pas attaquer
Brûlure	Divise l'attaque par 2 Le Pokémon perd 1/16 de PV entre chaque tour
Sommeil	Dure 1 à 3 tours Le Pokémon ne peut pas attaquer
Gel	Le Pokémon a une probabilité de 20% de dégeler à chaque tour Le Pokémon ne peut pas attaquer
Confusion	Le Pokémon a une probabilité de 33% de s'infliger des dégâts au lieu d'attaquer Dure 1 à 4 tours
Poison	Le Pokémon perd $\frac{1}{8}$ de PV entre chaque tour Si le Pokémon est gravement empoisonné alors il perd $\frac{k}{16}$ PV après le k-ème tour

Élément Attaque

Un Pokémon possède au plus 4 attaques distincts. Chaque attaque a

- un nom
- un id
- une puissance
- une précision : la précision varie de 0 à 101. Une attaque avec une précision de 60 a une probabilité de 40 % d'échouer. Une attaque avec une précision de 101 ne peut pas échouer.
- des points pouvoir (PP) : lorsqu'une attaque n'a plus de PP, elle ne peut plus être utilisée
- un type parmi Électrique, Psy, Poison, Spectre, Eau, Glace, Sol, Roche, Combat, Acier, Ténèbres, Feu, Plante, Insecte, Normal, Vol, Dragon, Fée. Un Pokémon utilisant une attaque du même type que le sien voit la puissance de son attaque multiplié par 1,5 (STAB).
- une catégorie : physique (1), spéciale (2) ou statut (3)
- une priorité : positive si l'attaque permet au Pokémon d'attaquer en premier, négative si elle permet au Pokémon d'attaquer en dernier. En cas d'égalité de priorité, c'est la vitesse du Pokémon

qui détermine lequel attaque en premier. En cas d'égalité de cette dernière, le premier Pokémon a attaquer est déterminé au hasard de manière équiprobable.

- une portée : nombre de Pokémon touché
- un court descriptif.

Élément Objet

Un Pokémon ne peut tenir qu'un objet au maximum. Un objet est obligatoirement tenu par un Pokémon. Chaque objet a

- un nom
- un id
- un descriptif
- des effets sur les statistiques du Pokémon et/ou son état. Certains objets peuvent aussi rajouter un type au Pokémon.

2.1.3 État général

A l'ensemble des éléments , nous rajoutons la propriétés suivantes :

- tour : représente le nombre de tour restant de modification de statistiques

2.2 Conception logiciel

Classe State : Les Pokémon en position 0 et 6 de la liste battle sont les Pokémon actifs, en avant sur le terrain. Seuls les Pokémon actifs peuvent attaquer. Les Pokémon en arrière ne peuvent rien faire, ils attendent d'être échangés.

La Classe State caractérise l'état du terrain. Elle est en bleu sur le diagramme.

Classe Pokémon : classe qui définit les caractéristiques d'un Pokémon. Elle contient un vecteur de 4 attaques et 1 objet.

Classe Attaque : classe qui définit les effets et caractéristiques d'une attaque

Classe Objet : classe qui définit les effets et caractéristiques d'un objet

Classe ModifStatsPokemon : classe qui permet de modifier les statistiques des Pokémon après un certains nombre de tours.

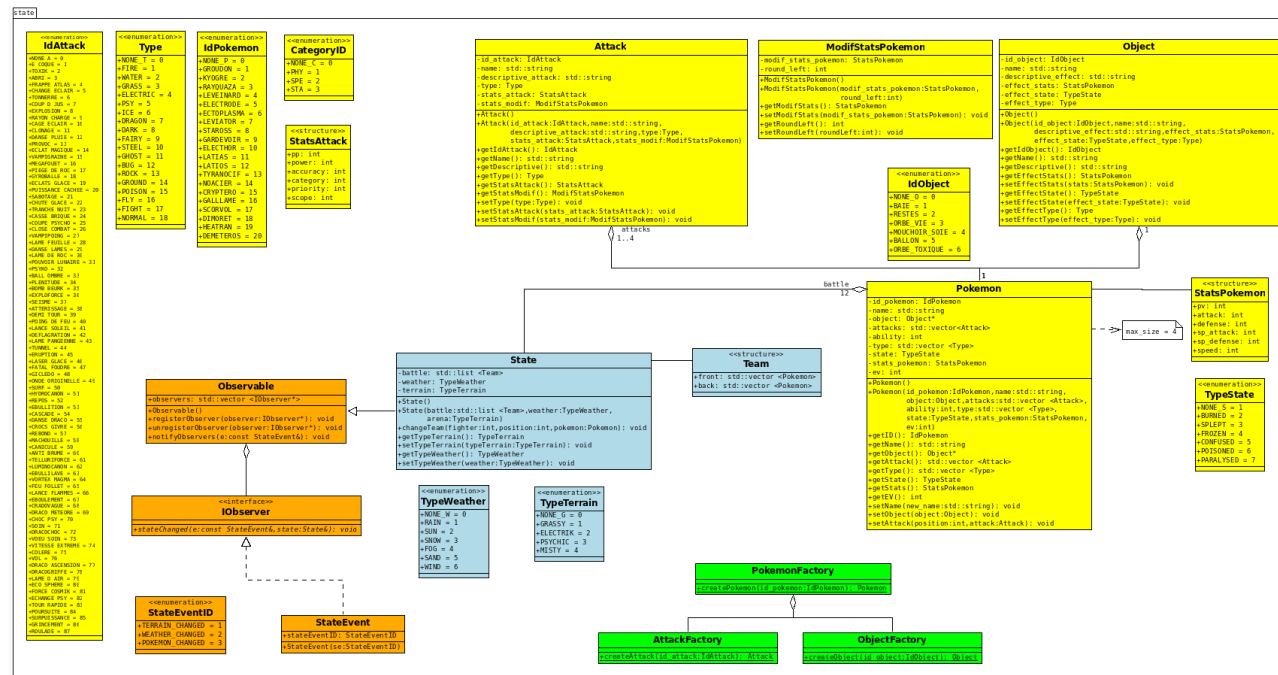
Ces classes permettent de caractériser l'état du joueur. Elles sont en jaunes sur le diagramme.

Classe Observable : cette classe notifie les changements d'états de State par exemple un changement de climat (weather) suite à une attaque lancée par un Pokémon ou un changement de Pokémon actif.

Classe PokemonFactory : classe possédant une méthode statique permettant de créer un pokémon sans avoir à instancier un objet au préalable. Elle ne prend en paramètre que l'Id du pokémon.

La classe AttackFactory et la classe ObjectFactory sont calquées sur la classes PokemonFactory, seul l'id en paramètre change.

Illustration 3: Diagramme des classes d'état



3 Rendu : Stratégie et Conception

Présentez ici la stratégie générale que vous comptez suivre pour rendre un état. Cela doit tenir compte des problématiques de synchronisation entre les changements d'états et la vitesse d'affichage à l'écran. Puis, lorsque vous serez rendu à la partie client/serveur, expliquez comment vous aller gérer les problèmes liés à la latence. Après cette description, présentez la conception logicielle. Pour celle-ci, il est fortement recommandé de former une première partie indépendante de toute librairie graphique, puis de présenter d'autres parties qui l'implémentent pour une librairie particulière. Enfin, toutes les classes de la première partie doivent avoir pour unique dépendance les classes d'état de la section précédente.

3.1 Stratégie de rendu d'un état

L'affichage est découpé en 3 parties : le background variant avec le climat et le terrain, les Pokémon actifs qui vont se superposer sur le background et l'interface permettant au joueur de choisir son action. Les sprites des backgrounds et Pokémon sont chargés en une seule fois, on effectue alors une lecture décalée pour changer le Pokémon ou background affiché. Cela améliore la rapidité du programme.

On utilise la librairie graphique SFML.

3.2 Conception logiciel

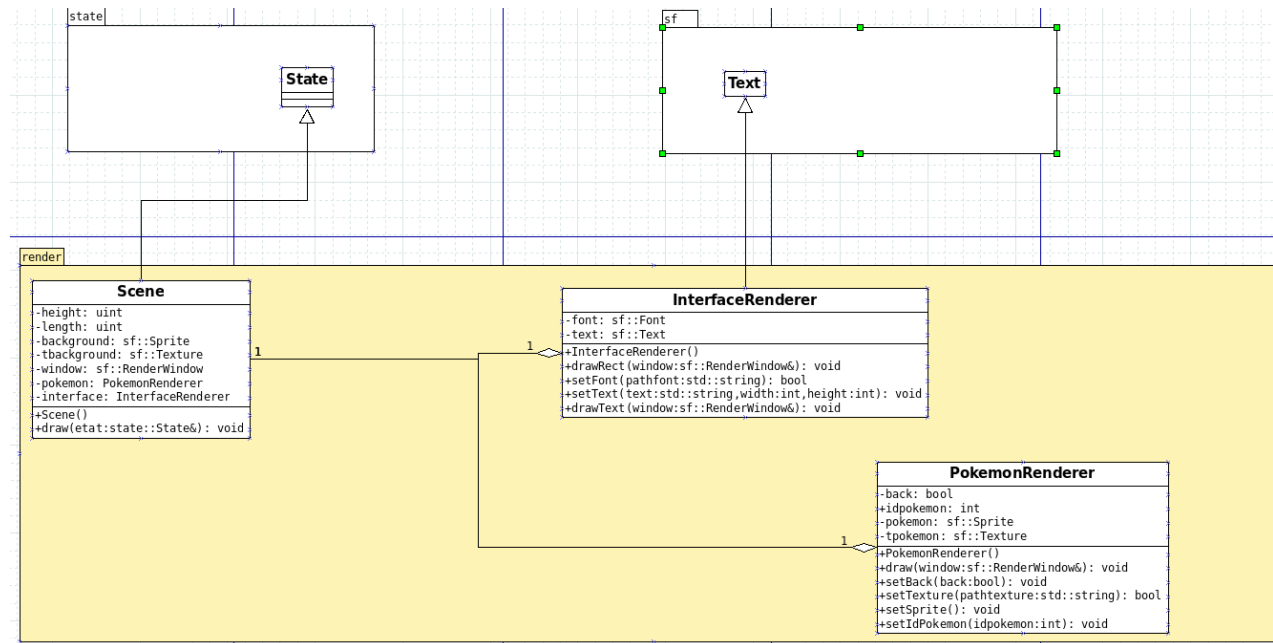
Classe Scene : Les attributs height et length permettent de fixer les dimensions de la fenêtre à afficher. La méthode draw() affiche la fenêtre et appelle les autres méthodes pour créer les textures et les sprites. Cette classe permet d'afficher les pokémon au combat.

Classe InterfaceRender : Cette classe permet l'affichage des choix du joueur et l'interface. Les attaques du Pokémon, avec les PP restants, totaux et le type, sont affichées à gauche. Les Pokémon de l'équipe sont affichés à droite. Si un Pokémon est KO, son sprite est plus sombre. Le Pokémon dont les attaques sont affichées est encadré grâce à la méthode drawSelect(). La méthode initPokemon() permet d'initialiser les sprites des Pokémon à afficher. La méthode initInterface() permet d'afficher l'interface. De plus si le joueur a déjà choisi son action pour un Pokémon, le message «OK» s'affiche sur le sprite de ce dernier.

Classe PokemonRender : La méthode draw() permet l'affichage du sprite du Pokémon de dos (pour le Pokémon du joueur) et de face (pour le Pokémon adverse). La position du Pokémon à afficher est récupérée depuis le vecteur battle dans le State. La méthode displayDamage() fait clignoter le Pokémon. Elle est utilisée lorsque le Pokémon a subi des dégâts.

Classe InfoRender : Cette classe possède toute l'information nécessaire à l'affichage de la barre d'information d'un Pokémon (nom, barre de vie, niveau et PV si c'est le Pokémon du joueur). La méthode displayDamage() permet de faire chuter la barre de vie du Pokémon ainsi que ses PV.

Illustration 4: Diagramme de classes pour le rendu



4 Règles de changement d'états et moteur de jeu

4.1 Changements extérieurs

Le joueur choisit, avec la souris, l'action qu'il veut effectuer pour chacun de ses deux Pokémon :

- une attaque parmi celle qui ont encore des PP,
- un pokémon parmi ceux qui ont encore des PV.

S'il choisi d'attaquer, il doit ensuite choisir sa cible avec les flèches directionnelles du clavier. Cependant certaines attaques ne cible pas l'adverse mais le Pokémon l'utilisant, dans ce cas le joueur n'a pas à choisir la cible.

Le joueur peut commencer par choisir l'action du premier ou du deuxième Pokémon. Pour cela il lui suffit de cliquer sur le Pokémon pour lequel il souhaite définir l'action. Il peut annuler son choix tant qu'il n'a pas choisi l'action des deux Pokémon.

L'IA va ensuite choisir son action suivant les mêmes règles.

Les actions sont classées par priorité, celle avec la plus haute priorité est exécutée en premier. En cas d'égalité, c'est le Pokémon avec la plus grande vitesse qui agit d'abord. S'il y a encore égalité, le premier à agir est décider au hasard.

4.2 Changements autonomes

Après le choix de l'action, il y a une première phase de changements autonomes :

- les PP des attaques utilisées sont décrémentés de un,
- les PV sont décrémentés en fonction des dégâts reçus,
- les Pokémon actifs sont actualisés, s'il y a eu échange.

En plus des changements autonomes se produisent après chaque tour :

- perte de PV si le temps inflige des dégâts ou si le Pokémon est affecté par le Poison ou la Brulûre
- récupération de PV due à un objet

Enfin on vérifie que chaque joueur possède au moins encore un Pokémon capable de se battre, c'est à dire avec des PV non nul. Sinon celui qui n'a plus de Pokémon capable de se battre a perdu.

4.3 Conception logiciel

Classe Engine : Cette classe possède un vecteur ordonnée de commandes à exécuter pour un tour. La méthode addCommand() permet d'ajouter une command en tenant compte de la priorité de l'action. La méthode runCommand() permet d'exécuter toutes les commandes de la liste, après l'exécution d'une commande, celle-ci est supprimée du vecteur. De plus la référence d'un vecteur est passée en paramètre. On y écrit l'ordre d'attaque des Pokémon afin d'afficher les animation dans le bon ordre. La commande undoCommand() permet de supprimer toutes les commandes du vecteur. Cela est utile lorsque le joueur souhaite changer son action.

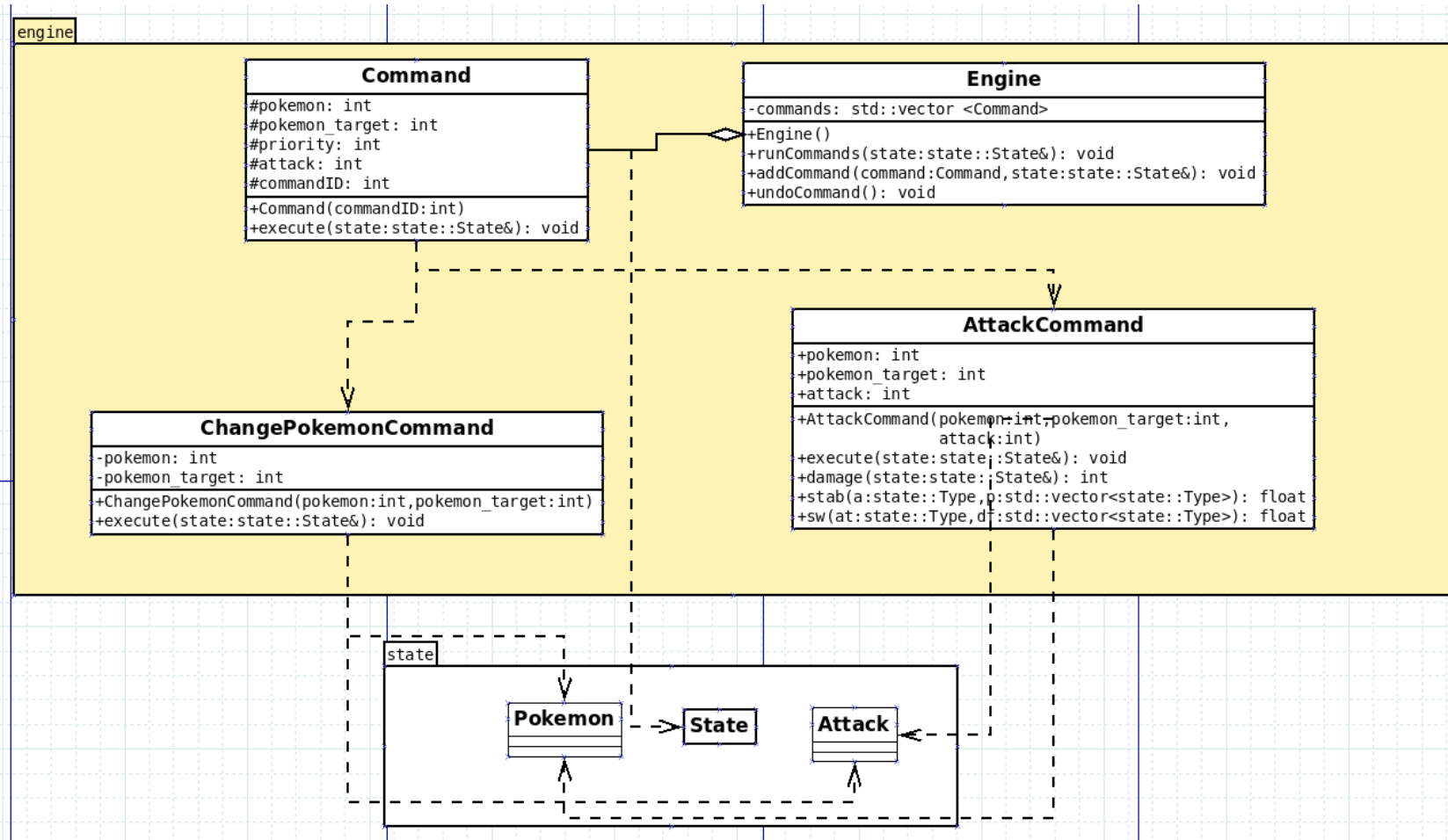
Classe ChangePokemonCommand : Cette classe permet d'échanger le pokemon à la position 'pokemon' avec celui à la position 'pokemon_target'. Toutes les informations sur les Pokémon sont échangées et le menu d'attaques est actualisé au prochain affichage.

Classe AttackCommand : Cette classe est appelée si le joueur ou l'IA choisit d'attaquer. Les

positions du pokémon attaquant (pokemon), du pokémon attaqué (pokemon_target) et de l'attaque utilisée (attack) sont initialisées. Les statistiques des pokémon sont utilisées pour calculer les dégâts tout en tenant compte des types et autres actions en effet pendant ce tour. Les PV et PP sont également mis à jour.

Classe Command : Cette classe dépend des classes ChangePokemonCommand et AttackCommand. Elle permet l'initialisation des attributs de ChangePokemonCommand et AttackCommand.

Illustration 5: Diagrammes des classes pour le moteur de jeu



5 Intelligence Artificielle

5.1 Stratégies

5.1.1 Intelligence minimale

Dans un premier temps, l'IA fera ses choix de manière aléatoire. Il choisira donc son action, attaque ou échange, et sa cible aléatoirement. Il ne tiendra pas compte des faiblesses et résistance et pourra donc lancer une attaque qui ne fait pas de dégât à un Pokémon.

5.1.2 Intelligence basée sur des heuristiques

Le but de l'IA est de mettre KO les Pokémon du joueur. La fonction heuristique estime le coût du chemin le plus court pour arriver au but. Donc l'IA doit maximiser les dégâts infligés à chaque tour par chaque Pokémon. Les échanges ne sont donc pas envisageable car il n'inflige pas de dégâts. Cependant l'IA peut être contraint d'échanger si le Pokémon s'il n'a plus de PP car il ne peut plus attaquer ou s'il est KO.

5.1.3 Intelligence basée sur les arbres de recherche

5.2 Conception logiciel : intelligence minimale

On utilise plusieurs fois la méthode rand() pour déterminer l'action des deux Pokémon de l'IA :

But	rand()
déterminer si le Pokémon va attaquer ou échanger	rand()%2
choisir le Pokémon cible	rand()%2 si c'est une attaque rand()%4+2 si c'est un échange
choisir l'attaque si nécessaire	rand()%4

5.3 Conception logiciel : extension pour l'IA composée

Pour chacune de ses attaques, l'IA va calculer les dégâts qu'il inflige à chacun des Pokémon du joueur sur le terrain et choisir la meilleure attaque. Les dégâts vont dépendre du type, de la catégorie et de la puissance de l'attaque ainsi que des statistiques du Pokémon attaquant et du Pokémon cible.

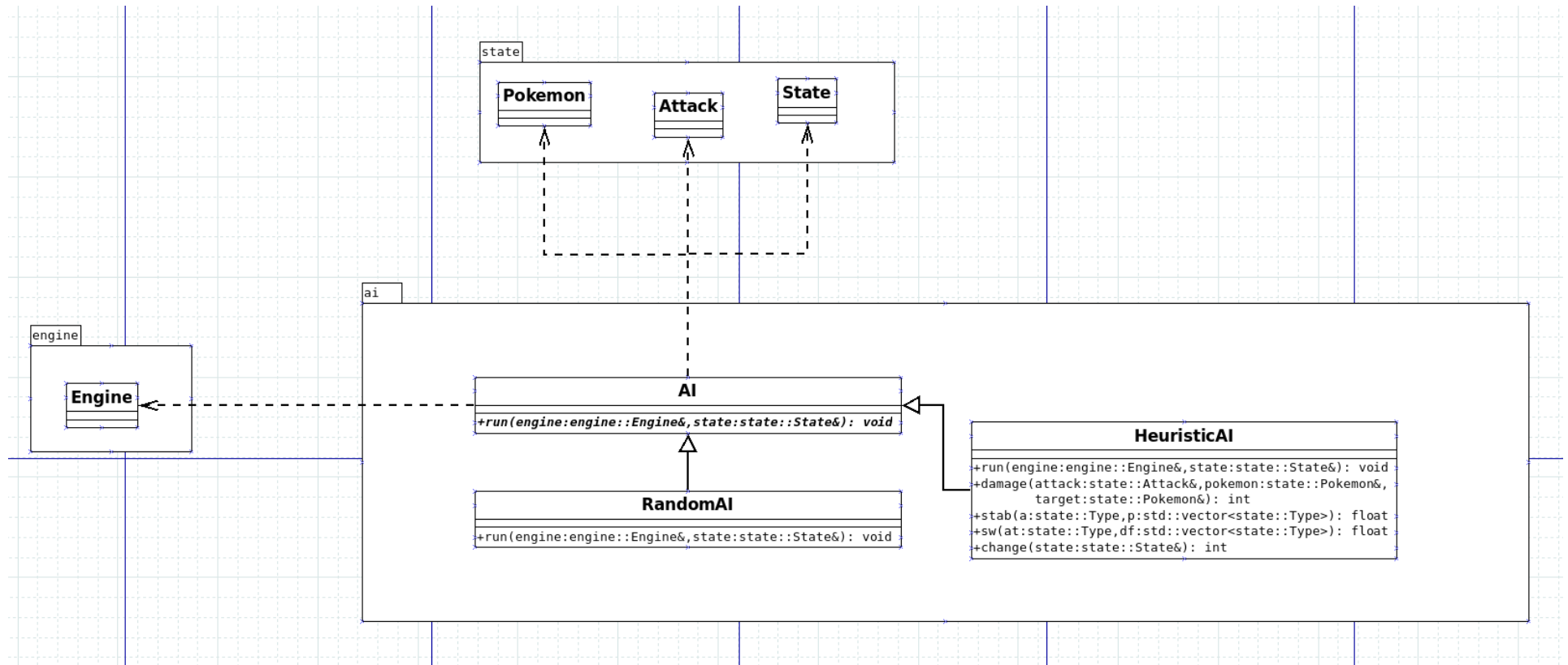
Si l'IA est obligé d'échanger l'un de ses Pokémon alors il va calculer les dégâts maximum de chaque attaque de chaque Pokémon qu'il lui reste et choisir le Pokémon qui fera le plus de dégâts.

Cette stratégie n'est pas la meilleure car elle ne tient pas compte des choix du joueur. Si celui-ci change de Pokémon alors tous les calculs de l'IA deviennent faux.

5.4 Conception logiciel : extension pour IA avancée

5.5 Conception logiciel : extension pour la parallélisation

Illustration 6: Diagrammes des classes pour le moteur de jeu



6 Modularisation

Cette section se concentre sur la répartition des différents modules du jeu dans différents processus. Deux niveaux doivent être considérés. Le premier est la répartition des modules sur différents threads. Notons bien que ce qui est attendu est une parallélisation maximale des traitements: il faut bien démontrer que l'intersection des processus communs ou bloquant est minimale. Le deuxième niveau est la répartition des modules sur différentes machines, via une interface réseau. Dans tous les cas, motivez vos choix, et indiquez également les latences qui en résulte.

6.1 Organisation des modules

6.1.1 Répartition sur différents threads

6.1.2 Répartition sur différentes machines

6.2 Conception logiciel

6.3 Conception logiciel : extension réseau

6.4 Conception logiciel : client Android

Illustration 7: Diagramme de classes pour la modularisation

