**Projet d’informatique : Combats de Pokémons**

**Descriptif du projet**

Mon projet est de faire un jeu de combats de Pokémons avec une intelligence artificielle contre laquelle on peut jouer. Un combat de Pokémon consiste dans l’affrontement de deux dresseurs possédant 6 Pokémons chacun (pas forcément les mêmes chacun) qui se battent entre eux 1 à 1. Le combat se déroule de la manière suivante : les dresseurs peuvent choisir une action chaque tour ; utiliser l’une des 4 attaques du pokémon combattant ou bien changer de pokémon combattant. Le gagnant est celui qui met K.O. tous les pokémons adverses (i.e. réduire leurs Points de Vie, PV, à 0). Le jeu est un peu comme un pierre-feuille-ciseaux en plus complexe ; les pokémons ont 1 ou 2 types chacun et chaque type est plus efficace contre des types de pokémon particuliers et faible face à d’autres : par exemple eau>feu, feu>plante et plante>eau. J’ai rentré le tableau des faiblesses dans la fonction « efficacite », il est difficilement lisible mais on peut le trouver sur [pokepedia](http://www.pokepedia.fr/Table_des_types#Tableau_g.C3.A9n.C3.A9ral). J’ai essayé dans les différents aspects de mon jeu de respecter le plus possible le jeu original Pokémon de Nintendo en exploitant notamment les données du site pokepedia.fr

**Comment jouer**

Il suffit d’exécuter tout le programme pour définir toutes les fonctions et quelques variables globales puis exécuter la commande « commencer\_partie() » ensuite il suffit uniquement d’entrer des chiffres pour faire les différents choix qui s’affichent dans la console, il n’y a pas d’interface graphique.

La commande « commencer\_partie("1","1",["1","2","3","4","5","6"],2,1,[-42]\*6,"1") » (à la ligne 893) permet de lancer directement une partie dont le premier joueur est humain, le deuxième est contrôlé par l’IA et chacun des joueurs possède les mêmes pokémons.

**Descriptif de quelques aspects du code**

J’ai créé plusieurs classes. La première, la classe « pokemon » qui prend plusieurs paramètres qui définissent le pokemon : ses statistiques, les différentes compétences qu’il connait, sa situation (PV actuels, endormi ou non, …). Cette classe a notamment une méthode « attaquer » qui permet d’utiliser une compétence (« competence » étant une autre classe) pour attaquer un autre pokémon. La puissance d’une attaque dépend de l’attaque utilisée, du type de l’attaque et du type du pokémon attaquant et du pokémon se faisant attaquer, des statistiques de chaque pokémon (attaque/défense), de leur état, d’éventuels bonus et du hasard. De plus une attaque peut rater et généralement une attaque très puissante aura une plus faible précision que les autres. De plus chaque attaque peut avoir un effet spécial (par exemple empoisonner l’adversaire ou bien modifier ses propres statistiques ou celles de l’adversaire). J’ai codé cela en mettant dans les paramètres de chaque compétence une fonction définissant son effet (ou None si elle n’a pas d’effet spécial) qui prend toujours les mêmes arguments en paramètres et que j’exécute à la fin de la fonction « attaquer ». Certains arguments sont alors parfois inutiles suivant l’effet.

J’ai également fait une classe « dresseur » qui représente le joueur. C’est le dresseur qui contrôle les pokémons qui leur donne les ordres. Le dresseur pourra être joué par un humain ou par une IA.

La fonction « tour » permet de faire un tour. Au début elle demande aux joueurs de prendre leurs décisions et ensuite le jeu exécute ces décisions suivant un ordre de priorité dépendant de la situation. En résumé, les changements de pokémons se font d’abord s’il y en a, puis le pokémon avec la vitesse la plus élevée utilise sa compétence. Si l’autre pokémon n’a pas été mis K.O. il peut utiliser sa compétence. Dès qu’un pokémon est mis K.O. le jeu demande au dresseur de choisir le prochain pokémon à envoyer au combat s’il lui en reste non K.O., sinon il a perdu. La fonction « commencer\_partie » demande au joueur de choisir les différents paramètres de la partie (joueurs/IA et pokémons) puis exécute la fonction tour tant qu’aucun joueur n’a perdu.

**L’intelligence artificielle**

J’avais d’abord commencé à programmer le jeu sans intelligence artificielle et j’avais fait en sorte que les différents choix puissent être faits avec la fonction « input ». Cependant cela pose problème avec une IA donc sous les conseils de Xavier Dupré j’ai fait une fonction « myinput » qui fonctionne comme la fonction input classique mais on peut lui rajouter un argument pour faire directement le choix sans avoir à rentrer quoi que ce soit après l’appel de la fonction. J’ai donc créé une IA qui répond à ces « myinput » toute seule.

Comment fonctionne l’IA ?

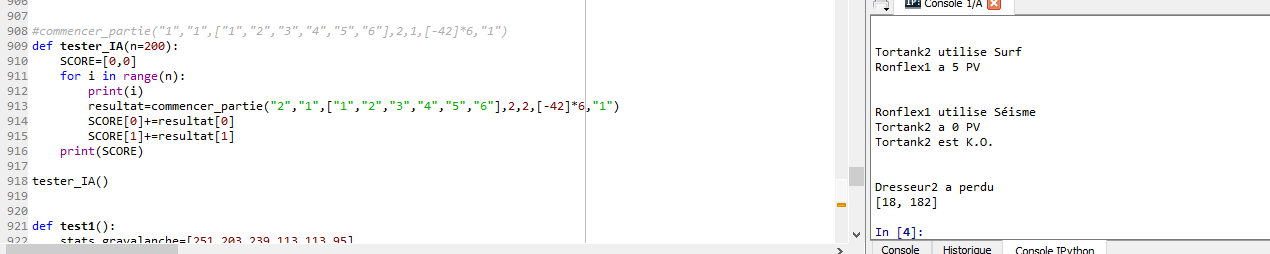
J’ai définit une fonction « score » qui détermine le score d’un pokemon. Ce score traduit l’utilité que peut apporter un pokemon. Un pokémon qui a tous ses PV sera vraisemblablement plus utile qu’un pokemon au bord du K.O. De même un pokémon endormit sera moins utile qu’un pokemon en pleine forme qui a reçu plusieurs bonus. La fonction score renvoie donc un nombre qui dépend de l’état de santé du pokémon.

Pour choisir quelle compétence utiliser, l’IA va tester toutes les possibilités et va déterminer la situation la plus favorable. Il y a 4²=16 possibilités puisque chaque pokémon connait 4 compétences - je n’ai pas considéré les changements de pokémons puisque l’IA n’anticipe qu’un seul tour alors que les changements de pokémons s’inscrivent dans une optique d’anticipation d’au moins deux tours, de plus les changements se font généralement dans des situations assez singulières ; ils sont rarement favorables puisqu’ils font perdre un tour d’attaque. La stratégie ressemble fortement à l’algorithme du Mini-Max : on va regarder les 4 décisions qu’on peut faire et les 4 décisions que peut prendre l’adversaire. Parmi les 4 décisions que l’on peut prendre, on va choisir celle qui va maximiser la différence des scores entre celui de son propre pokémon et celui du pokémon adverse si l’adversaire prend à chaque fois la décision qui minimise cette différence en sachant notre décision. Cependant, comme il y a une part d’aléatoire dans le résultat de chaque attaque, on ne peut pas savoir de manière déterministe la situation dans chacune des 16 possibilités de décisions. On va donc tester chaque possibilité plusieurs fois afin de pallier à cette incertitude.

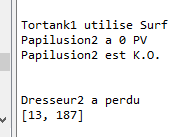
Il y a cependant un petit bémol. Cette stratégie n’est pas applicable dans le jeu original Pokémon puisqu’elle suppose une information parfaite des deux pokémons. On peut connaitre effectivement les différentes compétences et statistiques de son pokémon mais normalement, il y a une part d’inconnue sur le pokémon adverse. On peut certes savoir son type (feu, eau, …) mais on ne peut pas savoir ses compétences (bien qu’on puisse avoir un ordre d’idée ; un pokémon de type eau n’aura pas l’attaque « lance flamme » par exemple) et on ne peut pas connaitre ses statistiques donc on ne peut pas savoir à l’avance les dégâts de ses propres attaques outre l’aléatoire (un joueur expérimenté peut avoir un ordre d’idée, il sait que tel ou tel pokemon est résistant ou est rapide mais même deux pokémons de la même espèce aux mêmes niveaux peuvent avoir des statistiques qui diffèrent un peu). Comme dans mon jeu on doit choisir parmi une liste fixée de pokémons j’ai décidé qu’on pouvait considérer comme connues ces données. Toutefois j’ai créé une fonction « rerandomiser\_sommeil\_confusion » car lorsqu’un pokemon s’endort ou est rendu confus, le jeu détermine combien de tour cet état va durer mais les joueurs ne le savent pas. Or, si l’IA teste les différentes possibilités en faisant des copies des pokémons et en faisant un tour artificiellement (ce qui est la façon dont j’ai codé l’IA), elle se rend compte si un pokémon endormit se réveille ou pas alors qu’elle n’est pas censée avoir cette information. J’ai donc réintroduit de l’aléatoire pour éviter ce problème.

Pour choisir le pokémon à envoyer lorsqu’un pokémon est mis K.O. l’IA utilise le même principe du Mini-Max. Pour chaque pokémon qu’on peut envoyer, l’IA va faire le même test et va maximiser le résultat du test suivant le pokémon choisi.

J’ai également créé une IA qui joue aléatoirement. J’ai fait jouer l’IA qui utilise la stratégie du Mini-Max contre celle qui joue aléatoirement et elle semble satisfaisante puisqu’elle gagne plus de 90% des matchs.

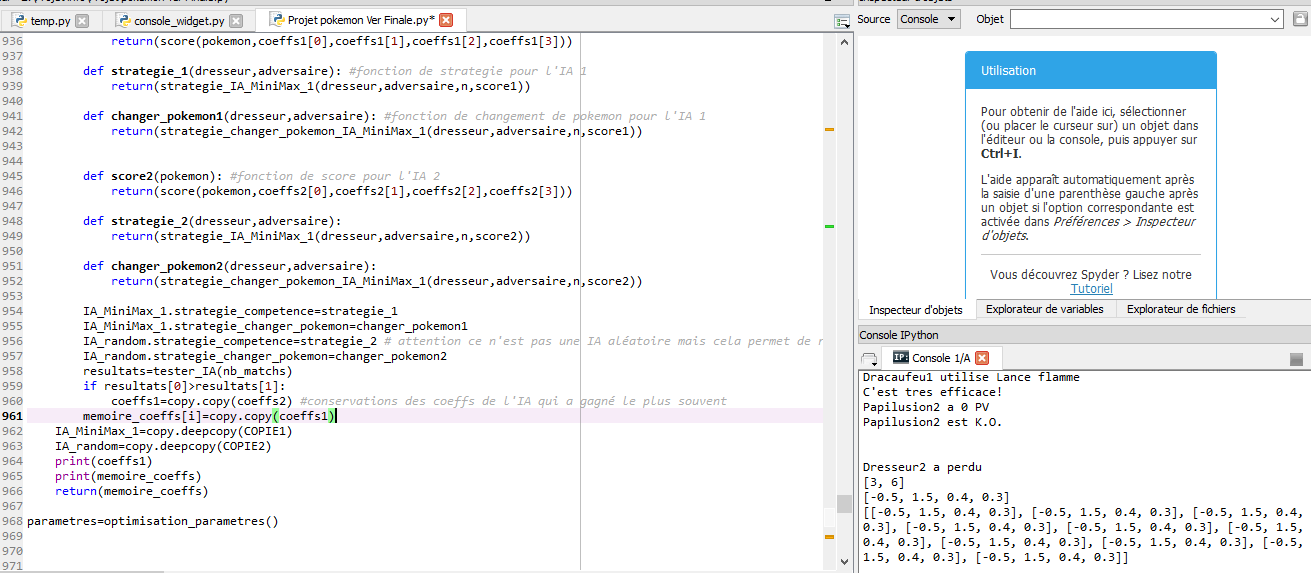


En testant 200 matchs, l’IA du MiniMax a perdu 18 matchs contre l’IA aléatoire qui a elle-même perdu 182 matchs. En refaisant 200 matchs le résultat est de 13 contre 187.

L’IA du MiniMax est donc clairement meilleure que quelqu’un qui joue aléatoirement. J’ai moi-même déjà perdu en jouant contre elle.

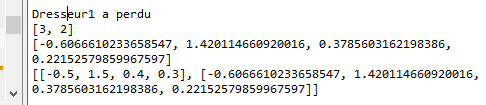
Toute la subtilité de l’IA est dans la définition du score d’un pokemon. En effet il n’est pas toujours très clair de déterminer si une situation est plus favorable qu’une autre. On peut donc penser que les différents paramètres définissant le calcul de ce score sont capitaux. J’ai au départ mis des coefficients au jugé en fonction de mon expérience du jeu. Ensuite j’ai créé une fonction « optimisation\_parametres » qui fait s’affronter deux IA utilisant la stratégie du Mini-Max mais pour la 2ème IA on modifie légèrement de manière aléatoire les paramètres. À la suite de plusieurs affrontement, on garde les paramètres de l’IA qui a le plus gagné et on recommence afin d’essayer d’obtenir la fonction de score la plus efficace possible.

J’ai bouclé ce processus 10 fois en faisant à chaque fois 9 matchs entre les 2 IA et finalement, les coefficients sont restés identiques à ceux que j’avais implémentés au début.

J’ai refait une 2ème fois 10 fois 9 matchs et les coefficients n’ont toujours pas bougé. Je me suis dit que soit la fonction score que j’ai implémentée était déjà très bonne ou bien il y avait une erreur. Finalement comme je le craignais il y avait une erreur.



Je me suis trompé dans la modification des coefficients. La liste de coefficients que je voulais modifier est restée intacte. Voici ce que j’obtiens en corrigeant ce problème avec 2 fois 5 matchs et des IA qui testent 3 fois chaque possibilité (je pensent que c’est insuffisant pour avoir une bonne optimisation mais cela prend beaucoup de temps)



**Ce que j’ai appris**

Durant mon projet, j’ai eu parfois du mal à détecter les erreurs car la plupart des fonctions appellent d’autres fonctions, parfois beaucoup d’autres fonctions. Il est parfois difficile de trouver où se trouve l’erreur. Heureusement que Python indique où il a rencontré l’erreur et indique toutes les fonctions qui ont été appelées avant qu’il arrive à l’erreur (en tout cas c’est ce qu’il fait avec Spyder). Cela m’a beaucoup servit. Cependant des fois Python ne renvoie pas d’erreurs car le problème est un petit bug qui fait que les fonctions ne font pas tout à fait ce que l’on veut mais elles ne déclenchent pas d’absurdité aux yeux de Python. L’un des exemples auxquels j’ai été confronté est la confusion entre « = » l’affectation et « == » le test. Lorsqu’on met une affectation au lieu du test, Python remarque tout de suite l’erreur et n’a même pas besoin de compiler pour l’indiquer. Cependant, il m’est arrivé, certes plus rarement, de faire l’erreur dans l’autre sens : mettre « == » alors que je voulais une affectation. Je me suis retrouvé ainsi dans la situation où une partie ne pouvait pas se terminer même si l’un des dresseurs avait tous ses pokémons K.O. simplement parce que dans ma fonction « verifier\_KO » j’ai mis « dresseur.perdu==True » au lieu de « dresseur.perdu=True ». Dans ce cas-là Python ne signale rien puisque il n’y a pas d’absurdité dans l’utilisation du langage. On se rend compte alors de la difficulté de résoudre les bugs sans l’assistance bien utile de Python.

**Et après ?**

Si on voulait poursuivre mon projet on pourrait d’une part ajouter d’autres pokémons puisque pour l’instant j’en ai créé que 7 ainsi que d’autres compétences. Cela pourrait diversifier le jeu. Par ailleurs, du côté algorithmique on pourrait essayer de refaire de l’optimisation de paramètre, qui prend beaucoup de temps (environ 1h30 pour faire 10 fois 9 matchs avec des IA qui testent 10 fois chaque possibilité). De plus j’ai traité de la même manière toutes les altérations d’état alors qu’elles ne se valent pas toutes. Par exemple la brûlure est clairement meilleure que l’empoisonnement. On pourrait donc introduire plus de paramètres, ainsi que prendre en compte la situation générale de la partie. Par exemple si on est en train de perdre (qu’on peut définir par la différence de la somme des scores des pokemons de chaque joueur) on peut valoriser la prise de risque c’est-à-dire la variance du score lorsqu’on teste plusieurs fois la même décision. On pourrait aussi faire une IA qui anticipe plusieurs tours. Cependant elle prendrait beaucoup de temps à prendre ses décisions. De plus il faudrait aussi changer les paramètres du calcul du score puisque tels qu’ils sont, ils prennent beaucoup en compte le futur avec une grande importance des bonus, de la confusion et des altérations d’état comme la paralysie. Or si on anticipe plusieurs tours, cette prise en compte du futur est déjà en partie incluse dans le fait qu’on tient compte de plusieurs tours.