Sorbonne Université

2I013 : Projet (application)

Sujet :

IA Football

Fangzhou Ye
Angelo Ortiz

Licence d'Informatique
Année 2017/2018

Table des matières

Introduction

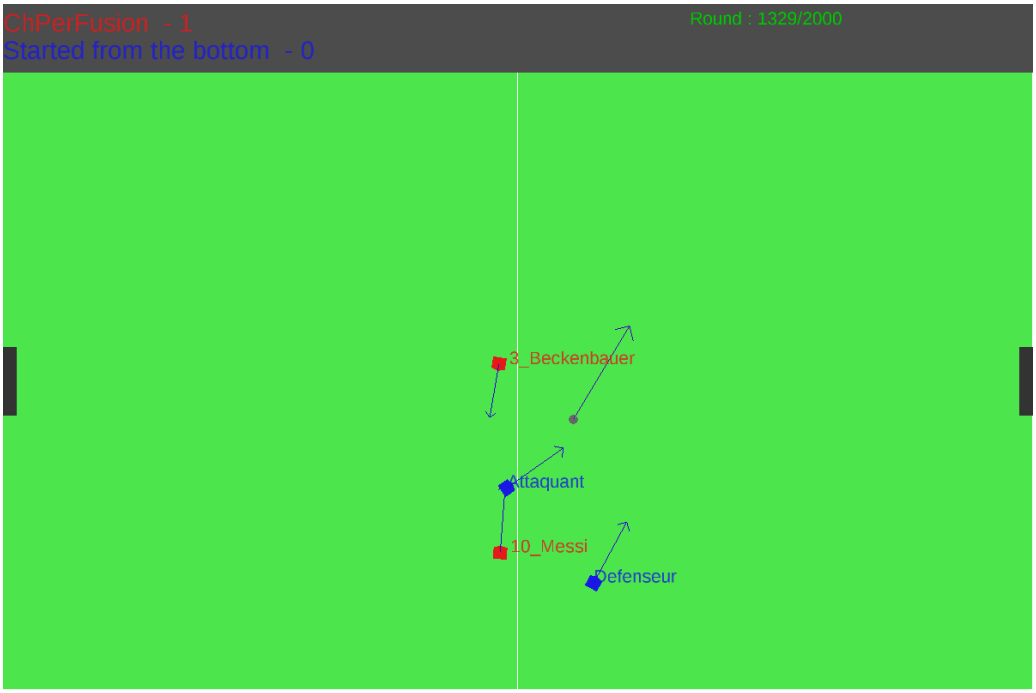
Présentation

Le projet a consisté en le développement d'intelligences artificielles de joueurs de football. Le simulateur du jeu étant déjà fourni, notre travail a été d'implémenter des stratégies de jeu pour mieux réussir les matches.

L'objectif du travail a été d'apprendre à bien mener un projet long sur un nouvel environnement de développement, à savoir Python. Sachant que travail à été fait en binôme, on a eu besoin d'un outil collaboratif, en l'occurrence **git** qui est un logiciel de gestion de versions décentralisé. On a ainsi appris à gérer l'avancement d'un projet à travers la méthode par fonctionnalités.

Aperçu

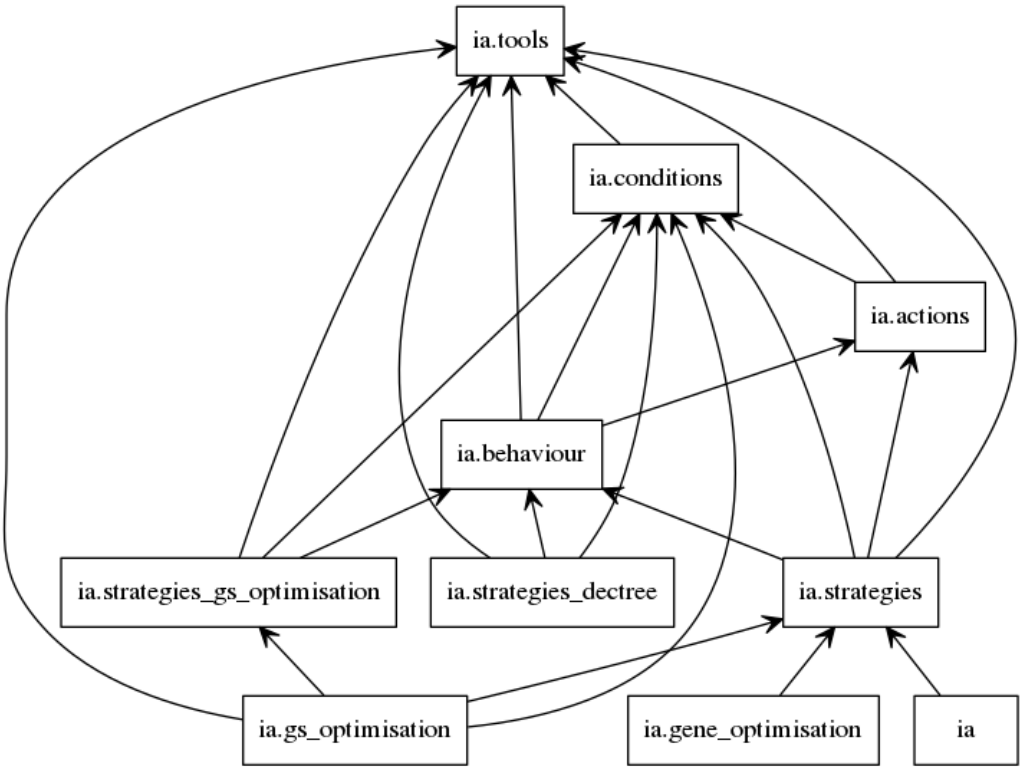
Voici un aperçu du simulateur de football lors d'une partie de notre équipe *ChPerFusion* en rouge face à l'équipe d'un de nos camarades de classe.



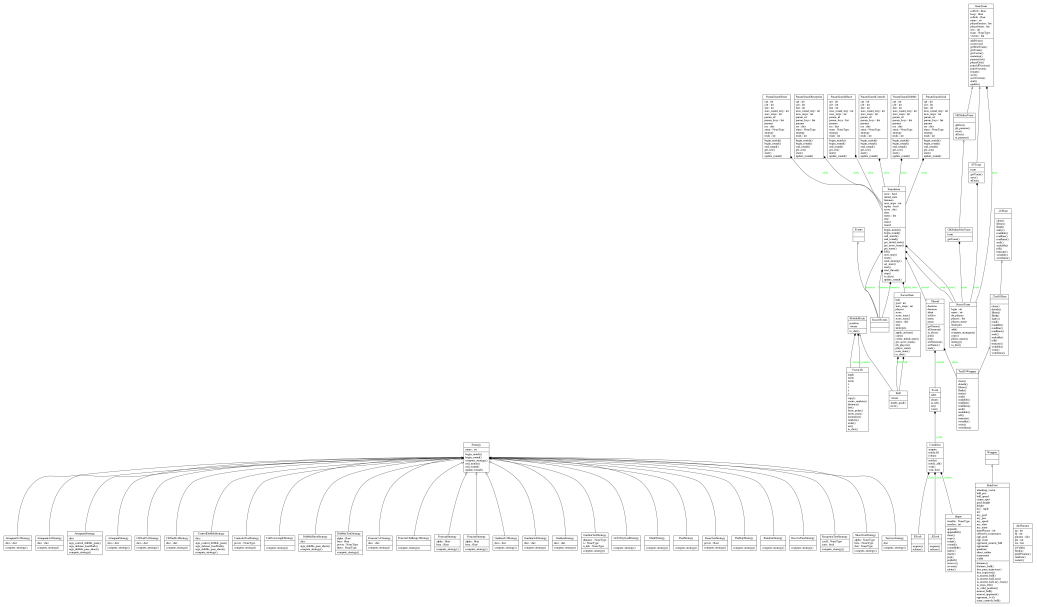
Diagrammes

Le projet a été organisé en un module **ia** qui contient l'implémentation nos joueurs et en divers fichiers de test.

On vous présente le graphe de dépendences du module.



On vous présente à présent le diagramme de classes du projet.



Première partie

Architecture logicielle

Dans cette première partie, on vous expliquera les choix faits concernant l'architecture logicielle.

Deuxième partie

Stratégies

Dans cette deuxième partie, on vous détaille le comportement de nos différents joueurs.

Compte tenu de la taille du terrain de jeu, le nombre maximum de joueurs par équipe est limité à quatre. Ainsi, il y a trois catégories de matches selon le nombre de joueurs par équipe : un, deux ou quatre.

1 Un joueur

Pour l'équipe composée uniquement d'un joueur, on a développé un seul joueur que l'on appelle *Fonceur*.

Fonceur

Au tout début du projet, il faisait tout au maximum : il doit s'approcher de la balle à toute vitesse lorsqu'il ne la contrôle pas et il frappe avec toute sa puissance dans le cas contraire. On a décidé de garder le même nom, même si son comportement diffère. En effet, à présent il avance avec la balle de petites distances et lorsqu'un adversaire lui fait opposition il essaie de le dribbler ; il frappe vers la cage adverse s'il se trouve dans la surface de réparation.

2 Deux joueurs

Cette catégorie a été celle sur laquelle on a consacré la plupart du projet. On compte trois jocusp pointueurs fonctionnels que l'on appelle *Attaquant*, *Gardien* et *Défenseur*.

Attaquant

Il a une vocation plutôt offensive. Il se comporte différemment selon s'il contrôle la balle ou pas. Dans la première situation, il essaie systématiquement d'avancer sur le terrain balle au pied et de faire un tir vers la cage dès qu'il se trouve dans la surface de réparation adverse. Cependant, s'il rencontre un adversaire lui bloquant le chemin, il tente de faire une passe vers son coéquipier s'il est sans marquage, il le dribble sinon.

Il montre trois conduites lorsqu'il se trouve sans le contrôle du ballon : il presse le porteur de la balle s'il en plus proche que son coéquipier, il se décale latéralement dans le cas contraire, et il monte dans le terrain si c'est son coéquipier qui la possède ou va bientôt en prendre possession pour lui proposer une possiblité de passe.

Gardien

De la même manière que pour le fonceur, le gardien ne garde guère de son comportement initial. Il restait fixé dans sa cage et en sortait juste pour réduire l'angle de frappe à l'attaquant adverse. Sa possession de la balle se réduisait à un dégagement presque aléatoire.

On a remplacé sa conduite avec la balle par celle de l'attaquant mentionné ci-dessus. En revanche, en absence du ballon, il reste à une distance de sa cage lorsque l'équipe adverse le contrôle et essaie de l'intercepter lorsque le porteur se trouve dans sa surface de réparation. Tout comme pour l'attaquant, il monte dans le terrain lorsque nécessaire.

Défenseur

Le défenseur n'a par contre qu'une vocation défensive. En effet, s'il a la balle, il s'en défait le plus rapidement possible soit à l'aide d'une passe vers son coéquipier si démarqué, soit via le dégagement le moins risqué possible. Quant à son effort défensif, il demeure à une certaine distance radial à partir de sa cage et s'approche de la balle en vu d'une interception s'il en suffisamment proche.

3 Quatre joueurs

Il s'agit de la catégorie que l'on a trouvé la plus intéressante en raison de la difficulté de la gestion des espaces et la répartition des fonctions. Tout comme pour la catégorie précedente, on compte trois joueurs fonctionnels que l'on repère d'ailleurs par les mêmes noms.

Attaquant

Il y a deux différences en son comportement par rapport à son pair de la catégorie ci-dessus et celles-ci concernent toutes les deux la situation où il se trouve dans sa surface de réparation. La première fait référence à la récupération de la balle : il ne se précipite plus et se comporte tel qu'il le fait dans le milieu du terrain. La deuxième correspond à l'effort de marquer les adversaires sans marquage.

Gardien

Ce joueur combine le comportement de son pair de la catégorie ci-dessus et de l'attaquant de cette catégorie. En effet, il suit les mêmes indications que l'attaquant lorsqu'il contrôle et réalise les mêmes mouvements sans ballon que le défenseur de l'équipe à deux joueurs.

Défenseur

Ce défenseur prend les mêmes décisions que celui de l'équipe à deux joueurs, à l'exception de sa prise de risque lors d'une interception. Effectivement, si un coéquipier se trouve dans une meilleure position pour intercepter la balle, i.e. il en est plus proche que le défenseur, celui-ci continue de positionner radialement à une certaine distance de sa cage.

Troisième partie

Méthodes d'optimisation

Dans cette troisième partie, on vous détaille les différents algorithmes appris tout au long du semestre pour l'amélioration des stratégies initialement proposées.

4 Recherche en grille

La première méthode d'optimisation preésentée en cours a été la recherche en grille. Celle-ci est utilisée pour l'optimisation d'une action.

Tout d'abord, on doit définir l'action à optimiser et le critère d'optimalité, et repérer les paramètres associés. Puis on obtient un ensemble de valeurs pour chaque paramètre. Pour les paramètres discrets, il suffit de prendre toutes les valeurs possibles, alors que pour les paramètres continus il faudra faire en amont une discrétisation, i.e. on divise l'intervalle délimité par des bornes définies selon un pas de précision. On compte ainsi des vecteurs dont les coordonnées correspondent chacune à un paramètre. On fait ensuite une recherche exhaustive sur tous les vecteurs obtenus : on teste chaque vecteur sous différents conditions environnementales et on moyenne les évaluations du critère dans lesdites conditions. Finalement, on prend la valeur optimale et son vecteur associé.

Il est important de remarquer que la plupart de nos paramètres sont continus. Pour obtenir des valeurs plus précises et un meilleur comportement, on a besoin d'un pas de discrétisation très petit. Ainsi, on réalise qu'une augmentation du nombre de paramètres à optimiser a une très forte influence sur le temps d'exécution de l'algorithme mis en place. On en déduit que le minuscule gain relatif nombre de paramètres–temps d'exécution ne convient pas au moyen terme.

5 Algorithmes génétiques

Dans un deuxième temps, on est passés aux algorithmes génétiques. On en a mis en place une implémentation pour optimiser les équipes composées d'un ou deux joueurs.

Cette classe d'algorithmes se base sur la théorie de l'évolution. En effet, la notion de sélection naturelle est le mécanisme permettant de choisir des solutions potentielles de plus en plus meilleures.

La toute première chose à faire est la définition de la fonction $f : \mathbb{R}^n \to \mathbb{E}$ à optimiser, où n est le nombre de paramètres concernés. Un élément $x = (x_1, \ldots, x_n) \in \mathbb{R}^n$ est appelé *candidat* ou *chromosome*, et chacune de ses coordonnées x_i avec $i = 1, \ldots, n$, appelée *propriété* ou encore *gène*, correspond à un paramètre à optimiser. Elles forment le *génotype*. Autrement dit, un candidat correspond à une équipe définie par les valeurs de ses propriétés.

Dans le cadre du projet, on a défini $f(x) = (V, N, D, P, C)$ comme le bilan après avoir disputé plusieurs matches avec les mêmes valeurs des paramètres données par x, i.e. le tuple composé du nombre de victoires, matches nuls, défaites, buts marqués et buts encaissés. La relation d'ordre suit les règles du classement d'un tournoi de football.

Cette classe d'algorithmes comportent quatre étapes.

1. Initialisation : Tout d'abord, il faut générer un ensemble de candidats $\{x_1, \ldots, x_m\}$. Le but est d'obtenir une population initiale la plus diverse possible de sorte à pouvoir atteindre le plus de maxima relatifs. C'est pourquoi cette première génération est obtenue de manière aléatoire.

2. Évaluation : Ensuite, on évalue la fonction en chacun des candidats, i.e. on calcule $\{f(x_1), \ldots, f(x_m)\}$.
3. Sélection : Suivant la notion de sélection naturelle, on ne conserve que les candidats avec les meilleurs résultats. Autrement dit, on dresse un classement des équipes associées aux candidats et n'en garde qu'une partie.
4. Reproduction : Finalement, on attribue les places libres aux candidats nés du brassage génétique des candidats les plus performants. Pour ce faire, on compte sur deux méthodes, à savoir le croisement et la mutation. Étant donné deux parents, on commence par les cloner en deux enfants. Puis on choisit une ligne de coupe divisant le génotype en deux parties. Ensuite, les enfants s'échangent une partie de leur génotype. Le croisement s'arrête à ce stade-là, tandis que la mutation ajoute du bruit aléatoirement dans un gène pour chacun des nouveaux chromosomes. On a décidé de faire une petite modification à cette étape-là lors de l'implémentation pour avoir plus de diversification : une minorité des places libres est attrribuée à des candidats générés aléatoirement.

Cet algorithme étant itératif, on recommence à la deuxième phase avec la nouvelle génération. Au bout d'un nombre raisonnable d'itérations, on se retrouve avec une génération composée de chromosomes très proches des maxima relatifs. On finit donc l'algorthme par prendre le candidat optimal de la dernière population.

6 Apprentissage automatique

Dans la dernière partie du semestre, on s'est concentré sur l'apprentissage automatique. On nous a présenté les deux grandes familles d'algorithmes d'apprentissage : supervisés et non supervisés, ainsi que l'apprentissage par renforcement. Deux méthodes ont été implémentées lors du projet.

Arbres de décision (de classification)

Il s'agit d'une technique d'apprentissage supervisé. Pour l'appliquer, on a besoin d'un espace de représentation $\mathcal{X}$, d'un ensemble d'étiquettes ou classes Y et d'une liste de m exemples d'apprentissage (x^i, y^i), où $x^i \in \mathcal{X}$, $y^i \in Y, i = 1, \ldots, m$, appelée ensemble d'apprentissage. Le but de cette technique est de trouver une fonction $f : \mathcal{X} \to Y$ telle que l'on puisse prédire à quelle classe appartient une future variable $x \in \mathcal{X}$.

Concrètement, on veut associer une certaine action à chaque état du jeu. Comme le nombre d'états possibles est trop grand, on utilisera une représentation par n caractéristiques généralisées qui précisent la configuration du terrain. On a ainsi que $\mathcal{X} = \mathbb{R}^n$, avec n la dimension de l'espace de représentation. De plus, l'ensemble d'étiquettes devient l'ensemble d'actions possibles A. On veut donc trouver une fonction $f : \mathbb{R}^n \to A$ qui fournit la meilleure action possible pour tout état du jeu.

Le problème auquel on est confrontés se réduit alors à partitionner l'espace de représentation en $p = |A|$ parties $\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_p$ de sorte que $f([\mathcal{X}_i]) = \{a_i\} \subset A$, $i = 1, \ldots, p$. On utilise les arbres de décisions pour représenter ce partionnement. Ceci implique que chaque nœud interne correspond à un test sur une des n dimensions de $\mathcal{X}$, chaque feuille correspond à l'action à entreprendre pour l'état donné par le chemin suivi depuis la racine.

On nous a fourni une stratégie spéciale `KeyboardStrategy` comportant les outils de base nécessaires à la mise en place de cette méthode, allant de la génération de l'ensemble d'apprentissage au partionnement de l'espace de représentation, en passant par l'application de la fonction lors des matches. Elle permet d'associer des touches clavier aux différentes stratégies de sorte que l'on puisse préciser la stratégie préconisée pour les différentes phases du jeu. Ceci

conforme ce que l'on appelle l'ensemble d'apprentissage. Cet ensemble est traité à l'aide des n caratéristiques généralisées définies auparavant et, à l'issue de ceci, on compte un partionnement de $\mathbb{R}^n$. Cette stratégie utilise alors la fonction f lors des matches suivants, ce qui donne un comportement plus intelligent à nos joueurs.

Pourquoi non utilisée ? touches

Q-learning

Cet algorithme d'apprentissage par renforcement répose sur un ensemble d'états S, un ensemble d'actions A, et une fonction $Q : S \times A \to \mathbb{R}$, appelée *politique*, pondérant une paire formée par un *état* donné et une action à entreprendre pour cet état-ci. Cette pondération est surnommée *récompense*. La notion d'état correspond à celle donnée dans la section des arbres binaires. Il utilise aussi la notion d'*épisode* que l'on associe à un match dans le cadre du projet. Son principe est le suivant : on apprend une politique comportementale qui optimise l'espérance des récompenses.

La première phase consiste à initialiser Q à une valeur fixe arbitraire, ici 0. La deuxième phase concerne un épisode, ou encore un match, et est répétée autant de fois qu'il y a d'épisodes. Pour un état s_t donné, on choisit une action a_t selon la politique Q et observe la récompense r_t associée à cette paire. Cette action génère un nouvel état s_{t+1}. Il est à noter qu'il n'y a pas de choix concernant l'état initial s_0, puisqu'il est toujours le même pour un match de football : ceci correspond au coup d'envoi. Ensuite, on met à jour la politique selon la formule ci-dessous.

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{valeur actuelle}} + \alpha \cdot \underbrace{\left(r_t + \gamma \cdot \max_a(Q_{t+1}, a)\right)}_{\text{valeur apprise}}$$

Ici, α représente la vitesse d'apprentissage et γ le facteur d'actualisation. D'un côté, la vitesse d'apprentissage détermine l'équilibre entre exploration et exploitation. En effet, une valeur de 0 implique l'utilisation exclusive des choix actuels, alors qu'une valeur 1 ne ferait que considérer le dernier résultat. D'un autre côté, le facteur d'actualisation détermine l'importance des récompenses futures. Une valeur de 0 ne prend en considération que les récompenses courantes, tandis qu'une valeur de 1 met en valeur les récompenses plus lointaines. On avait précisé qu'un épisode correspondait à un match ; par conséquent, la réussite d'une succession d'actions dépend du résultat final du match. De ce fait, on a intérêt à retarder au maximum les récompenses. Ceci se traduit par le choix d'un facteur γ proche de 1.

Juste après la mise à jour de la politique Q, on détermine si l'état s_t est terminal, i.e. qu'il correspond au coup de sifflet final. Dans le cas affirmatif, on passe à l'épisode suivant ou on s'arrête selon s'il y en a encore à faire, alors que l'on réitère cette phase à partir du choix d'une action pour le nouvel état s_{t+1} dans le cas contraire.

À la fin de l'algorithme, on se retrouve avec une politique qui nous rend l'action la plus adéquate pour tout état du jeu. Étant donné que les caratéristiques utilisées pour définir un état sont continues, on a défini des intervalles d'équivalence. Cela n'empêche que le nombre total d'états est énorme et fait donc exploser la mémoire. On pourrait mettre en place un réseau de neurones pour pallier cette difficulté. Ceci n'a pas été le cas pour ce projet et donc, bien qu'implémenté, on s'est pas servi de cet algorithme pour améliorer nos stratégies.

Conclusion

...