16/05/2015

Football et stratégie

Programmation d'intelligence artificielle pour joueurs de foot
Rapport de projet – 2i013

André Nasturas - 3201101

Table des matières

Football et stratégie

Rapport de projet – 2i013

Étudiant en seconde année de licence informatique à l'Université Pierre et Marie Curie, j'ai eu à choisir un projet parmi les modules d'enseignement. Intéressé par l'intelligence artificielle, mon choix s'est porté sur le projet *Football et Stratégie*.

Ce projet consiste à programmer des joueurs de football virtuels, animés dans un simulateur fourni, dans le but de les rendre les plus performants possible. Il s'agit donc de coder une intelligence artificielle efficace pour une équipe de joueurs.

Quelle est la meilleure stratégie possible pour ces joueurs, et comment l'implémenter ? Peut-on créer un joueur de foot virtuel réellement intelligent ?

L'environnement de travail

Tous les codes sources sont stockés et *versionnés* sur **GitHub**[1]. Cela s'avère très pratique, entre autres pour coder avec d'autres ordinateurs que ceux de l'université. De plus, l'utilisation de git comme base de développement apporte plusieurs avantages comme une meilleure organisation, le versionnement des fichiers (et donc la possibilité d'annuler des modifications), ou encore la possibilité de créer plusieurs branches afin de développer plusieurs fonctionnalités en parallèle.

Pour développer en Python, il suffit d'un éditeur de texte et d'une console Python. Le terminal pourrait donc faire l'affaire, mais n'est pas très confortable. Spyder est un éditeur complet avec une console python intégré, il est donc très pratique.

J'utilise également **Brackets**[2], un éditeur de code esthétique et efficace qui ressemble au célèbre **Sublime Text**[3]. La gestion native de Python, la sauvegarde automatique et

l'intégration de Git à l'interface rendent le développement sur ce logiciel rapide et agréable.

Principe

Nous commençons avec un simulateur simplifié de football fourni, codé en Python. Il s'agit de développer des stratégies pour des joueurs de foot – en héritant et modifiant convenablement un objet *SoccerStrategy* fourni vide dans le code du simulateur, qui seront éprouvés lors de matchs entre les équipes des différents développeurs du projet.

Action

Une action d'un joueur est représentée par la classe *SoccerAction*, et comporte deux informations

- Un vecteur de **mouvement** indiquant la direction et l'accélération avec lequel va se déplacer le joueur.
- Un vecteur de **tir** correspondant à la direction et à la force appliquée à la balle par le joueur (si la balle est à portée de tir, évidemment).

A chaque instant, un joueur va effectuer une action consistant donc à courir dans une direction et éventuellement frapper la balle. La stratégie que l'on va coder devra donc déterminer, à chaque instant, quelle action mener.

Stratégie

La stratégie de chaque joueur, codée dans un objet héritant de la classe *SoccerStrategy*, est appelée régulièrement pendant la partie par le simulateur. En fonction de certaines données sur le jeu, elle permet de générer une action qui sera appliquée par le simulateur à ce joueur. Les données disponibles pour appliquer la stratégie sont les suivantes :

- Le joueur, de classe *SoccerPlayer*, contient toutes les informations courantes à propos du joueur actif, notamment sa position sur le terrain et sa possibilité de tirer (un joueur ne pouvant pas tirer plusieurs fois de suite sans délai).
- L'état de la partie (*SoccerState*). C'est ici que l'on trouve toutes les informations sur la partie : la liste des joueurs et équipes en jeu, le score, la position et l'état de la balle…
- Le teamid, ou le numéro d'équipe (qui détermine si un joueur doit marquer à gauche ou à droite).

Outils

A ces éléments fournis par le simulateur de foot, on peut ajouter une boîte à outils personnalisée : *SoccerTool*. Il s'agit d'une mini-bibliothèque de fonctions qui facilite la programmation des stratégies. Par exemple, on peut coder une fonction *goToGoal()* qui retourne directement donnant la direction du but cible par rapport au joueur actif.

Technique

Lors de chaque unité de temps, le simulateur appelle la fonction *compute_strategy()* de la stratégie de chaque joueur, en lui passant comme argument les informations sur la partie. Cette fonction doit renvoyer un objet de type *SoccerAction*, soit un vecteur de déplacement et un vecteur de tir.

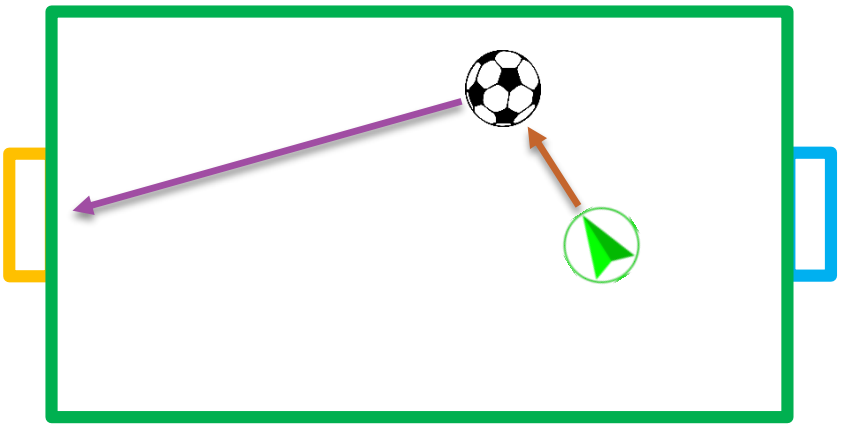
C'est donc cette fonction qui accaparera la majorité du travail de ce projet.

Premières tactiques

On commence par coder quelques premières stratégies basiques. En voici quelques exemples.

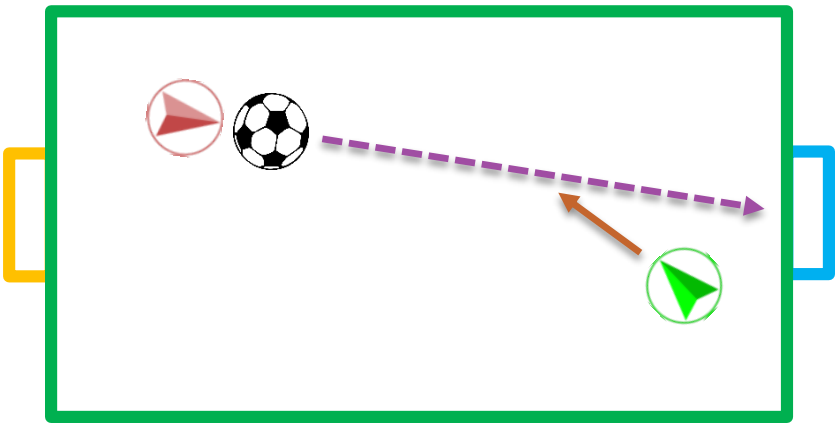
Coureur

Une première stratégie simpliste consiste à simplement courir vers la balle pour la frapper vers les cages ennemies. Autrement dit, à chaque instant, le joueur va se déplacer à vitesse maximale vers la position actuelle de la balle et frapper celle-ci si possible dans la direction des cages adverses.



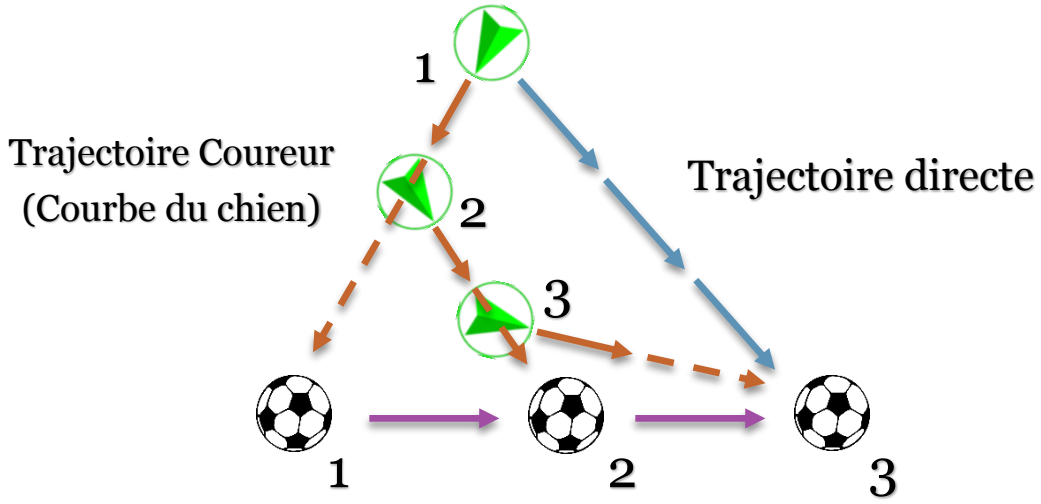
Défenseur

Le joueur va rester entre la balle est les cages à défendre, pour l'intercepter en cas d'attaque ennemie.



Intercepteur

C'est une variante du défenseur : le joueur va tenter de récupérer la balle le plus rapidement possible en tenant compte du déplacement de la balle et de sa décélération. L'idée est d'atteindre plus rapidement la balle qu'avec le Coureur en évitant de perdre du temps sur une trajectoire de forme courbe de chien[4] (flèches brunes).



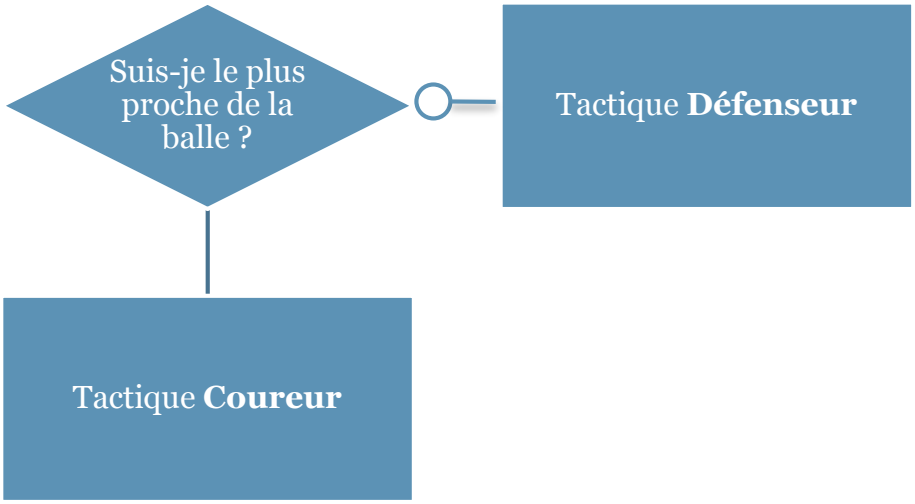
Cependant, pour obtenir une trajectoire réellement aussi directe que possible, il faut aussi prendre en compte la physique du jeu appliquée par le simulateur, et notamment l'inertie des joueurs lorsqu'ils changent de direction.

Un début d'intelligence

Grâce aux tactiques élémentaires établies précédemment, il est déjà possible de jouer des matchs, et même d'en gagner. Il est par exemple possible de composer une équipe de deux coureurs et deux défenseurs.

Conditions et actions

Mais des rôles fixés à l'avances et immuables ne permettent pas un jeu très efficace ou sophistiqué. Pour cela, il faut conditionner l'usage de ces tactiques dans une stratégie plus globale, qui choisira l'action à effectuer en fonction de la situation sur le terrain, grâce à un arbre de décision. Par exemple, une stratégie simple mais adaptative est illustré par l'organigramme décisionnel ci-dessous : si le joueur est plus proche de la balle que tous les autres joueurs (information qui peut être obtenue grâce à une fonction-outil de *SoccerTool*), il courra vers la balle. Sinon, il se placera en position défensive.



Il n'est donc pas forcément nécessaire de coder une stratégie excessivement complexe pour obtenir une solution efficace. On peut, à la place, utiliser plusieurs tactiques élémentaires vues précédemment tels que courir, se démarquer, défendre ou passer. Puis, il faut identifier les situations où il est le plus opportun d'appliquer l'une ou l'autre de ces tactiques : lorsque le joueur est à proximité de la balle, des cages ou d'un ennemi ; lorsqu'un allié est démarqué, ou en position de marquer un but ; lorsque l'ennemi contrôle la balle, etc…

Apprentissage

Cependant, identifier et décrire toutes les situations où il est indiqué d'appliquer telle ou telle stratégie n'est pas évident. Pour cela, il y a l'apprentissage. Il s'agit d'un code spécial, ajouté au simulateur de matchs, qui va enregistrer à chaque instant l'état de la partie et la stratégie utilisée par chaque joueur. Associé à un joueur *interactif*, c'est-à-dire dont on peut manuellement changer la stratégie active à tout moment, cela permet de générer des données brutes. Le code d'apprentissage se chargeant ensuite de filtrer ces données pour garder les variables pertinentes – celles dont une variation significative est signe qu'il faut changer de stratégie – et de générer le nouvel arbre de décision.

Grâce à cela, il devient possible de réaliser une intelligence artificielle capable d'interpréter à un certain niveau la situation et décider d'appliquer une stratégie adaptée en conséquence. Il est même envisageable d'automatiser complètement le processus et de laisser l'IA tester divers cas et apprendre toute seule.

Conclusion

Dans le cadre du projet du module 2i013, j'ai eu à programmer une intelligence artificielle capable d'animer un joueur de football virtuel et de le faire gagner.

Une première étape a consisté à découvrir le langage Python utilisé dans ce projet, un langage dont les bases sont similaires aux autres langages célèbres, mais doté d'une syntaxe légèrement différente.

Il m'a aussi fallu maîtriser git, le code étant versionné sur GitHub. Cela ne m'a pas été très difficile, ayant déjà eu à utiliser git et subversion auparavant.

Après cela, j'ai codé quelques tactiques de base, en assimilant le fonctionnement de SoccerSimulator.

J'ai ensuite dû améliorer mon code, en créant des fonctions-outils adaptés et en normalisant les classes de stratégies pour les rendre utilisables avec un sélecteur, en vue de la conception d'une stratégie à arbre de décision et de l'apprentissage.

J'ai ainsi obtenu une stratégie relativement efficace, en ayant en même temps découvert un nouveau langage, mais aussi appris à développer un projet à l'aide d'un gestionnaire de versions et basé sur un autre projet extérieur.

Glossaire

1. **GitHub** (*github.com*) est un service web d'hébergement et de gestion de dépôts git.
2. **Brackets** (*brackets.io*) est un éditeur de code open-source proposé par Adobe.
3. **Sublime Text** (*sublimetext.com*) est un puissant éditeur de code payant.
4. **Courbe du chien** (*fr.wikipedia.org/wiki/Courbe_du_chien*) est la courbe décrite par un objet orientant constamment sa trajectoire dans la direction d'une cible mouvante (courbe de poursuite).