2I013 - Projet

Groupe 1 – Football et Stratégie

Elodie GIANG – Lisa LAM

2017 - 2018

Sommaire

1. Introduction

Ce projet se présente comme un jeu de football simplifié : deux équipes s'affrontent sur un terrain conçu de façon à ce que la balle rebondisse lorsqu'elle touche les bords du terrain. Le temps de jeu a été discrétisé. A chaque pas, le joueur a accès à la position de chaque élément du terrain, que ce soit son coéquipier, son adversaire ou la balle. Il effectue alors une action en fonction de la stratégie choisie.

Le but est de programmer des stratégies afin d'obtenir les joueurs les plus performants possibles et d'être le mieux classé lors des tournois se déroulant chaque semaine dans lesquels chaque binôme confrontent leurs joueurs.

Dans ce rapport, nous présenterons notre approche du projet et expliquerons au fur et à mesure les choix effectués et les problèmes rencontrés.

2. Structure du code

Afin de faciliter l'écriture des stratégies et généraliser les utilisations récurrentes de fonctions et de tests telles que si le joueur peut tirer ou non ou plus simplement encore la position de la balle, nous avons regroupé ces méthodes dans deux classes distinctes.

2.1. Tools

La classe tools contient les fonctions récurrentes liées à l'environnement du joueur. Elle a été créée afin de simplifier le code et le rendre ainsi plus lisible. Elle prend state, id_team et id_player en argument afin de ne pas avoir à les réécrire à chaque utilisation d'une des fonctions et ne pas ainsi alourdir le code. Pour faciliter l'utilisation de ces fonctions, nous avons veillé à ne pas avoir de problème de symétrie ie à ne pas avoir à se soucier de l'id_team du joueur.
Elle contient les fonctions renvoyant :

- la position des principaux éléments : le joueur, la balle, les cages adverses...
- le vecteur ou la distance entre les différents éléments : vecteur entre la balle et les cages adverse, entre le joueur et la balle, le joueur et ses cages…
- un booléen selon le respect ou non d'une condition : si la balle est dans la moitié / le tiers / le quart du terrain, si l'adversaire est à l'intérieur du périmètre défini autour du joueur (**Figure 1**), si l'adversaire est devant le joueur (**Figure 2**)…
- un joueur particulier : l'adversaire ou le coéquipier le plus proche

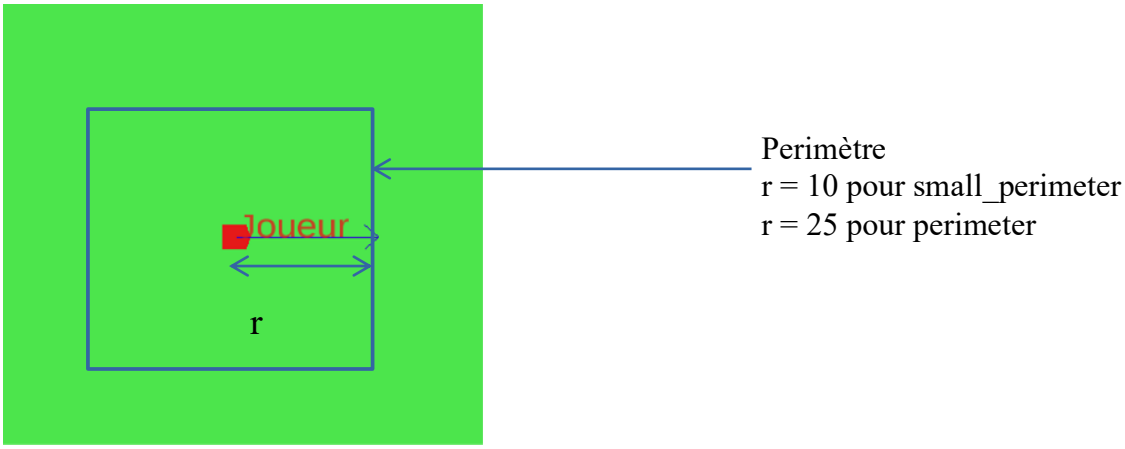


Figure 1 : schéma du perimètre défini dans certaines fonctions de la classe tools (ennemi_in_my_perimeter, ennemi_in_my_small_perimeter, ball_in_my_perimeter…)

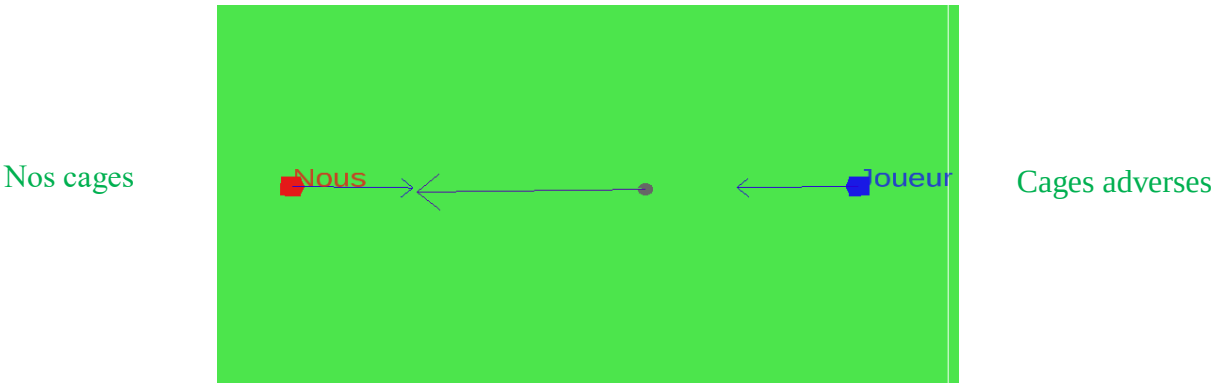


Figure 2 : le joueur adverse (en bleu) est devant notre joueur (en rouge) car il est entre ses cages et notre joueur

2.2. Comportement

La classe Comportement permet de définir les fonctions qui déterminent l'action du joueur a partir de l'état, de l'identité du joueur et de son équipe.
Définissons certaines fonctions utiles au projet :

- Les fonctions shoot, run, run_anticipe, passe permettent de définir la vitesse et direction de la trajectoire du joueur et le vecteur accélération de la balle. Respectivement, elles permettent aux joueurs de se diriger vers les cages et de tirer vers les cages, courir vers le ballon, courir vers le ballon en anticipant sa trajectoire et shooter vers le joueur de son équipe le plus proche.
- Les fonctions se_demarquer/marquer_joueur permettent de s'éloigner/se rapprocher du joueur adverse le plus proche.
- Pour spécifier un peu plus les positions de défense dont nous avions besoin pour les différentes stratégies de défense, nous avons défini des fonctions return_goal, return_defense, return_defense_top/down/milieu et return_attaquant_defense qui permettent aux défenseurs ou attaquants de retourner respectivement vers son goal, retourner en position de défense, de se placer respectivement en haut/bas/milieu de son milieu de terrain, de retourner à une distance de 1/3 de son goal (**Figure 4**).
- La fonction degage permet de tirer dans le ballon vers les cages adverses avec un decalage de + ou – 30 en fonction de la position du joueur adverse le plus proche.
- La fonction dribble (**Figure 3**), tout comme la fonction dégage, utilise un décalage de + ou -50 en fonction de la position du joueur adverse le plus proche pour dribbler ce joueur adverse.
- Les fonctions petit_shoot, shoot2 et shoot3 sont une variation de la fonction shoot. Le premier atténue la puissance du shoot, le second utilise une fonction g définie dans la classe tools, et le dernier prend en paramètre la puissance strength. Cette dernière fonction sera utile pour la recherche d'optimisation.
- La fonction dribble2 est une variation de la fonction dribble, qui prend en paramètre la puissance strength et le décalage decal afin d'optimiser ces deux paramètres grace à la recherche d'optimisation.
- Enfin, la fonction follow_fonceur est utilisée par le défenseur dans le but de suivre les mouvements en x du fonceur, avec un décalage prédéfini.

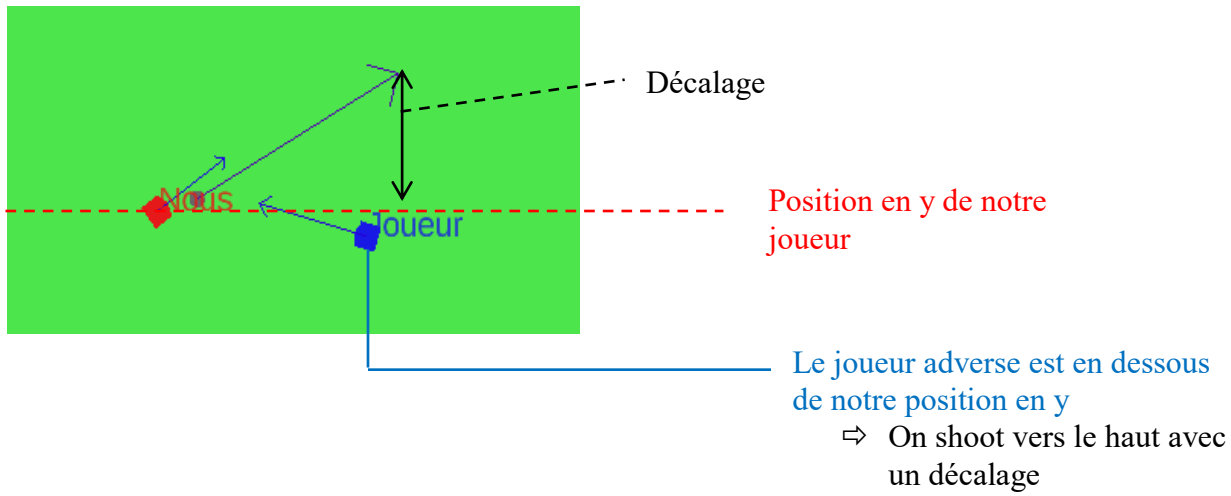


Figure 3 : Schéma du fonctionnement du dribble

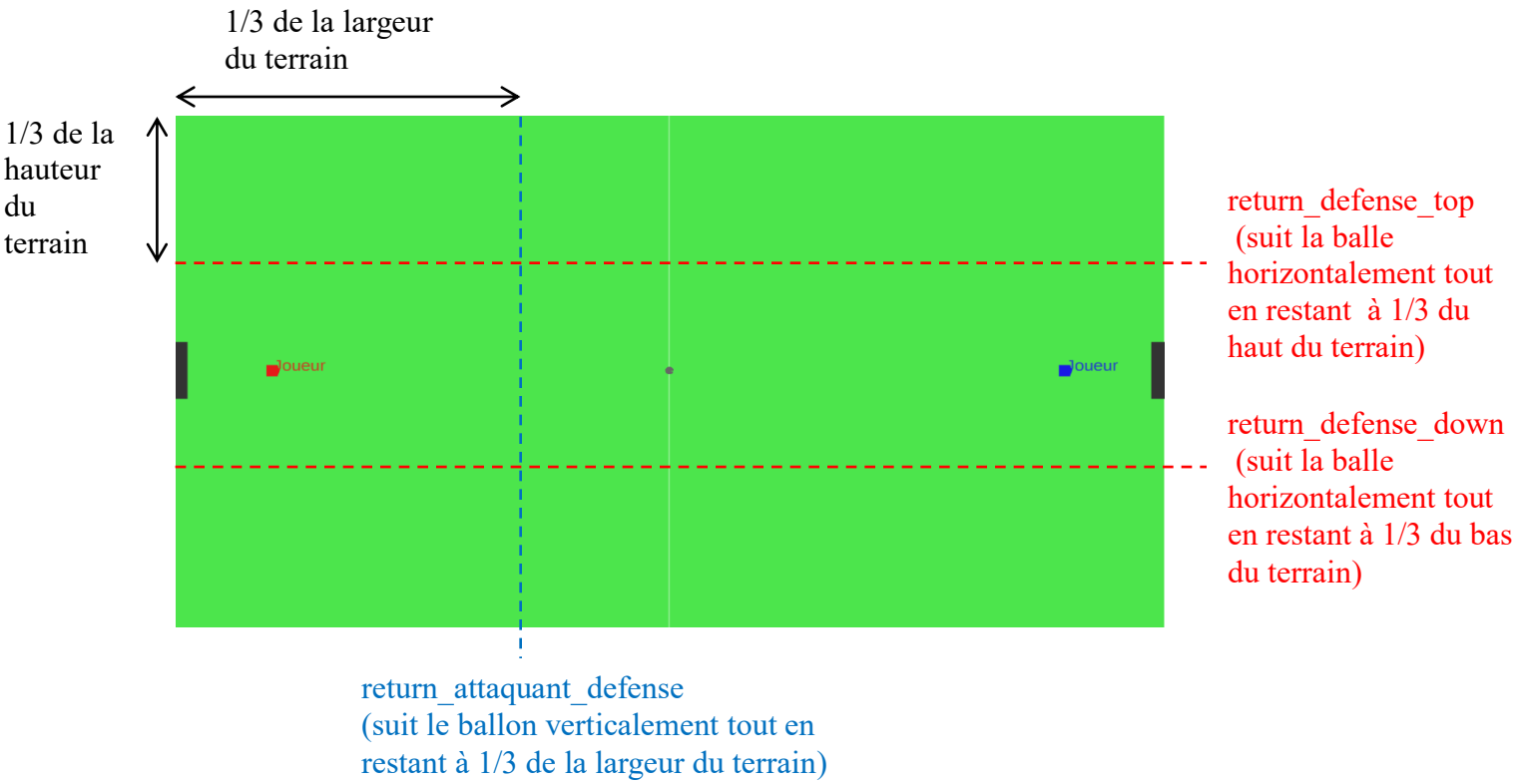


Figure 4 : Schéma des positions de défense (return_attaquant_defense, return_defense_top/down)

3. Stratégies

Nous avons défini plusieurs types de stratégies attaquantes ou défensives selon le nombre de joueurs placés sur le terrain.

3.1. Fonceur3

Il s'agit de notre attaquant pour le 1v1 (**Figure 5**).

Nous avons défini cette stratégie pour que le joueur attende cent pas de temps avant de foncer vers la balle afin d'éviter le plus possible la confrontation avec le joueur adverse lors de l'engagement, ce qui facilitait la récupération du ballon. En effet, cela permettait d'éviter un comportement trop prévisible et de mieux anticiper les mouvements de l'adversaire et ainsi avoir l'avantage. Cette attente est rendue possible grâce à un compteur ajouté dans la classe de cette statégie. Il se réinitialise lorsqu'une des deux équipes a marqué et qu'une nouvelle partie commence.
Cependant, si le joueur adverse se met en mouvement et amène la balle dans notre moitié de terrain, notre joueur se mettra à son tour en action de la même manière que défini ci-dessous après les cent pas de temps.
Après les cent pas de temps :

- Si le joueur est assez proche du ballon pour pouvoir shooter, il va dribbler ou shooter selon son environnement :
  - Si l'adversaire est devant lui et est très proche il va dribbler avec une forte puissance de shoot afin d'éviter que l'adversaire ne lui prenne le ballon car quand l'adversaire est trop proche, la priorité est de défendre. Dans le cas contraire, il va dribbler avec une faible

puissance afin de ne pas être trop éloigné du ballon mais pouvoir tout de même perturber le joueur adverse.

- Si l'adversaire se trouve derrière lui et qu'il se trouve autour des cages adverses, il va shooter vers les cages : on a en effet les conditions optimales pour qu'il puisse marquer. Sinon il va faire un shoot avec une faible puissance en direction des cages. La puissance est faible pour qu'il puisse avancer dans sa progression sans que le ballon ne parte dans tous les sens à cause de l'aléa présent dans le shoot.

- Si le joueur ne peut pas shooter, il va foncer vers le ballon en anticipant sa trajectoire.

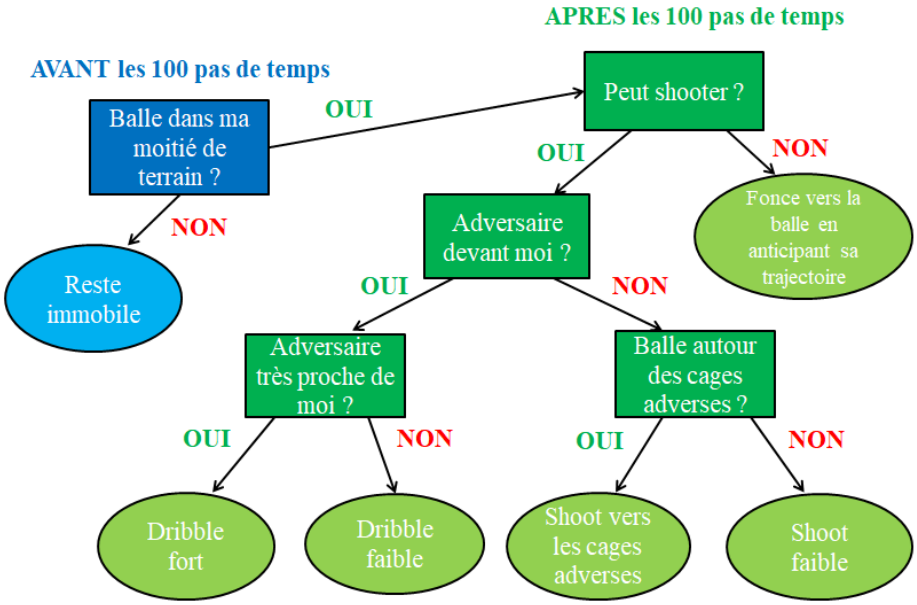


Figure 5 : Schéma représentant les actions du Fonceur3

3.2. Fonceur3_2v2

Il s'agit de notre attaquant pour le 2v2.

Il agit comme le Fonceur3, mais prend en compte son coéquipier et lui fait des passes. La passe est effectuée seulement si un il y a un joueur adverse devant, qu'un joueur adverse soit très proche de lui mais qu'il n'a pas de joueur adverse qui soit très proche de son coéquipier, et que son coéquipier ne soit pas trop loin de lui. Son coéquipier ne doit pas être trop loin sinon, lors de la passe, la balle ne l'atteindra pas et cela faciliterait la récupération de la balle par l'équipe adverse.

3.3. Fonceur3_top, Fonceur3_down

Il s'agit de nos deux attaquants pour le 4v4.

Ils agissent sur le même principe que le Fonceur3 expliqué précédemment, mais sont plus adaptés au 4v4 car ils prennent en compte qu'ils sont plusieurs joueurs dans l'équipe. Ils se répartissent le terrain : le Fonceur3_top n'agit que sur le haut du terrain tandis que le Fonceur3_down agit sur le bas du terrain. Cela permet de mieux posséder l'espace du terrain et d'éviter que les deux attaquants ne fassent les mêmes actions au même moment.

Si un adversaire est trop proche du Fonceur3_top et qu'il n'y a pas d'ennemi autour du Fonceur3_down, alors le Fonceur3_top va lui faire une passe. Cela vaut également dans le cas inverse.

Dans le cas où le ballon n'est pas dans le haut (resp. bas) du terrain, le Fonceur3_top (resp. Fonceur3_down) se contente de suivre le ballon afin d'être toujours proche de l'action pour venir en aide à son coéquipier.

3.4. Defenseur_2v2

Pour notre défenseur 2v2 (**Figure 6**), nous avons défini un compteur, représentant les pas, qui a chaque gain de point, se réinitialisera à 0. Ainsi, le défenseur aura un comportement qui différera en fonction des pas de temps.

Si la balle se trouve dans la moitié du terrain du défenseur, si le défenseur peut shooter la balle, et s'il n'y a pas d'ennemi autour de son allié le plus proche, il lui fera la passe. Si au contraire l'allié le plus proche est marqué par un joueur adverse, le défenseur dégagera la balle. Si le défenseur n'est pas encore à la portée du ballon, il anticipera la trajectoire du ballon et ira le réceptionner.

Cette stratégie permet au défenseur d'anticiper les prises de ballon de l'adversaire. En effet, en testant si la voie entre le joueur et le défenseur est libre, le défenseur limite les prises de ballon de l'adversaire. Si un ennemi franchit la moitié du terrain du défenseur, celui-ci prendra immédiatement place en défense. Si le ballon est situé dans la moitié de terrain de l'adversaire, si tous les adversaires sont situés derrière le défenseur et que le joueur se situe dans un périmètre prédéfini autour du goal adverse, il shootera, car il n'aura plus d'obstacle entre lui-même et le goal. Cependant si quelques ennemis se trouvent entre lui et leur goal, il fera une passe s'il voit un allié démarqué, sinon il dribblera. Ici, grâce à cette méthode de jeu, le défenseur évite le plus possible les trajectoires de balle qui pourra être interceptées par un ennemi.Si les conditions plus haut ne sont pas remplies, le défenseur retournera en défense.

La subtilité est qu'avant 100 pas de temps, le défenseur retournera en défense. Au delà, il suivra le fonceur en x avec un décalage de 15 prédéfini.

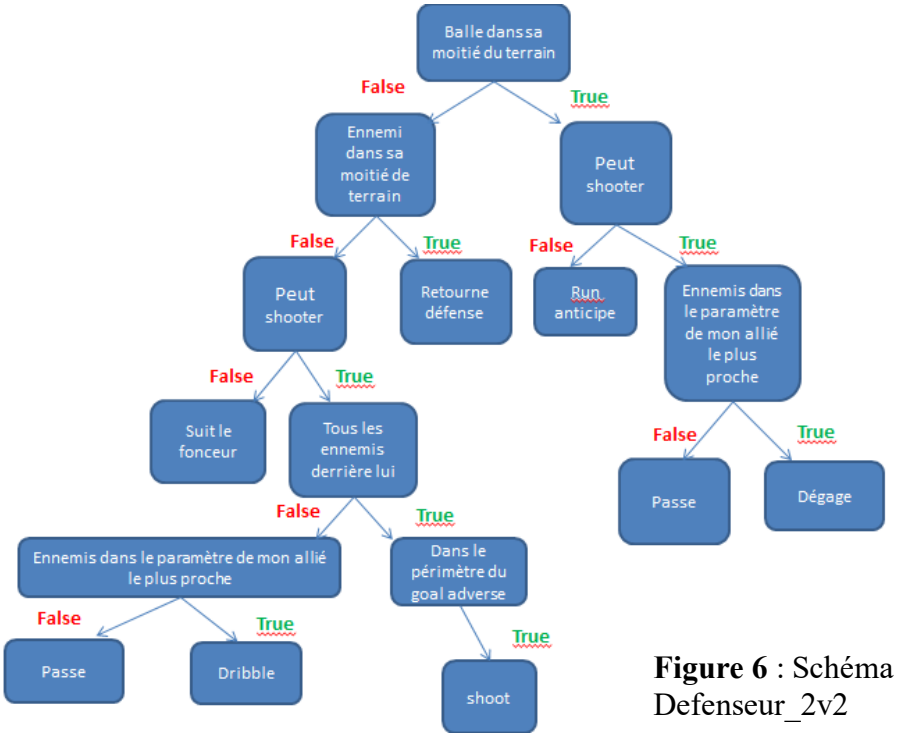


Figure 6 : Schéma représentant les actions du Defenseur_2v2

3.5. Defenseur_top, Defenseur_down

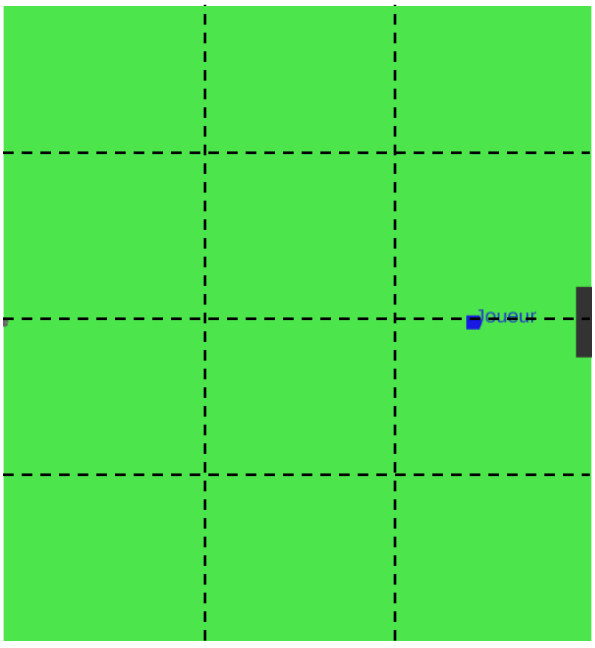
Pour le cas 4v4, nous avons deux types de défenseurs. Un Defenseur_down, situé au début de la partie, vers le bas de son milieu de terrain, et un Defenseur_top, situé quant-à lui vers le haut de son milieu de terrain. Les deux défenses sont quasi similaires :

- Si le ballon est situé dans son tiers du terrain, et s'il peut shooter, il dégagera la balle. S'il ne peut pas shooter, il courra vers la balle en anticipant sa trajectoire.
- Si elle n'est pas située dans son tiers de terrain, mais si elle est située dans la moitié haute du terrain, alors le Defenseur_top retournera à sa position initiale, en haut de son milieu de terrain, et le Denfenseur_down en bas de son milieu de terrain. Dans les cas contraires, les défenseurs se placeront au milieu du terrain. Ces différents placements selon la position du ballon permettent aux défenseurs de gérer une partie du terrain, et de ne pas se gêner dans les mouvements, ou faire la même chose, ce qui serait inutile. Au contraire, nous essayons d'optimiser au mieux les deux défenseurs pour qu'ils soient complémentaires l'un de l'autre et qu'ils puissent défendre l'un, la moitié haute du terrain, et l'autre, la moitié basse du terrain.

4. Optimisation

Au départ, pour choisir les valeurs telles que la puissance du shoot ou du dribble, nous devions faire des tests à la main ce qui était long, fastidieux et ne nous assurait pas d'avoir le meilleur paramétrage possible. C'est pourquoi nous avons eu recours à des méthodes d'optimisation.

4.1. Recherche en grille



Le but de l'optimisation en grille est de séparer le terrain en plusieurs cases (**Figure 7**) afin d'optimiser les paramètres optimisables pour chaque carré de la grille. Pour cela, on modifie la fonction begin_round de la classe optimization afin de choisir dans quel carré de terrain le ballon apparaîtra en début de step. La position du joueur sera de même celle du ballon. En appelant ParamSearch, on obtient un dictionnaire de résultats nous permettant de déterminer quelle est la meilleure valeur pour ce paramètre. Nous avons écrit en plus, quelques lignes de codes qui nous permettront de trier ces résultats et afficher directement la valeur optimale.

Figure 7 : Décomposition du terrain en une grille 4x3 : on lance ParamSearch pour chaque carré

4.2. Algorithme génétique

Le procédé d'optimisation par l'algorithme génétique se déroule en quatre temps :

- On génère aléatoirement des candidats en fonction des paramètres devant être optimisés.
- On effectue les calculs sur ces candidats qui nous permettent d'obtenir un résultat en fonction d'un critère choisi.
- On sélectionne les candidats ayant eu le meilleur potentiel, ie ceux qui ont obtenu le critère le plus élevé.
- On génère la nouvelle génération en fonction des candidats choisis précédemment. Pour cela on réalise deux mécanismes de brassage génétique : l'enjambement, qui échange certains paramètres entre deux candidats, et la mutation, qui modifie un paramètre d'un candidat.

Nous avons utilisé l'algorithme génétique afin d'optimiser le dribble. Les paramètres sont donc la force du dribble, son décalage et le rayon du périmètre à partir duquel, si un adversaire y pénètre, le joueur va dribbler. Le critère est le nombre de buts marqués. On créer des candidats représentés par un tuple contenant les paramètres dont les valeurs ont été choisies aléatoirement. On appelle ParamSearch sur chaque candidat puis on ne garde que les meilleurs auxquels on applique les fonctions d'enjambement et de mutation. On recommence jusqu'à n'avoir plus qu'un seul candidat.

Les résultats obtenus n'ont pas été satisfaisants : les valeurs que nous avions choisies à la main étaient plus performantes. Cela pourrait s'expliquer par un nombre insuffisant de candidats créés et un intervalle de valeurs trop grand pour le choix des valeurs aléatoires des paramètres.

4.3. Arbre de décision

L'arbre de décision nous permet de visualiser quelles sont les meilleures stratégies à appliquer à un joueur en particulier en fonction de situations de jeu spécifiques.

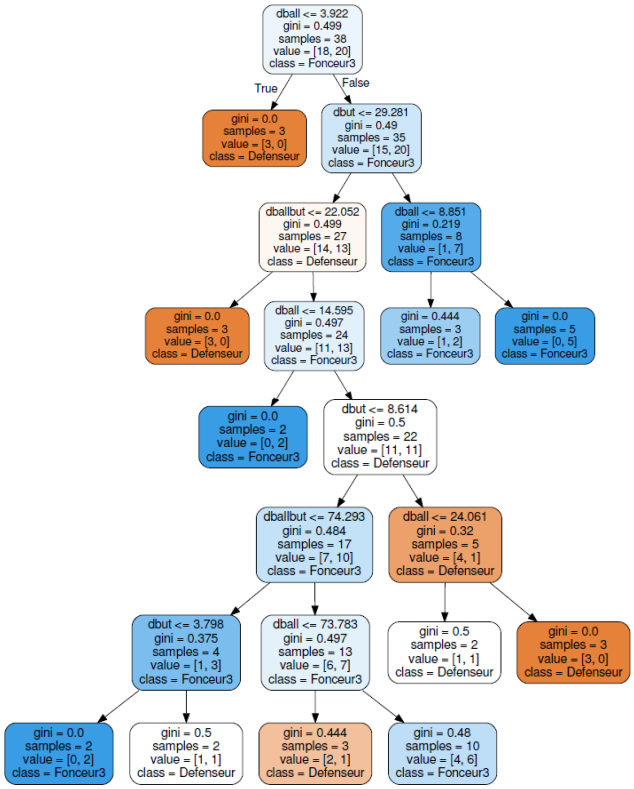
Pour ce faire, nous avons utilisé le fichier exemple_apprentissage.py que nous avons modifié afin d'appliquer toutes les stratégies que nous avons définies. Pour commencer, nous avons choisi des vecteurs qui influeront sur le choix de la stratégie à appliquer selon les situations de jeu : la distance à la balle, la distance au but, la distance balle/but, la distance joueur/but et la distance ennemi/joueur. Il reste ensuite à choisir les stratégies à appliquer au joueur en fonction de la touche de clavier appuyée. Pour cela, on doit créer une partie et ajouter ces joueurs à cette partie. Par exemple, en écrivant « kb_strat.add("a",Fonceur3Strategy()) », nous avons ajouté le joueur Fonceur3Strategy, qui sera effectif seulement si la touche « a » est enfoncée.

En exécutant dans la console, on affiche une partie entre deux joueurs. Un joueur que l'on peut contrôler grâce aux touches prédéfinies plus haut, et un joueur fonceur (nous avons décidé de faire jouer notre joueur contrôlé contre un Fonceur3Strategy.

A la fin de notre simulation, nous obtenons un arbre de décision (**Figure 8**) qui permet de voir la stratégie la mieux adaptée pour différents types de situations.

Cependant, nous avons rencontrés quelques problèmes. En effet, nous avions eu du mal a contrôler notre joueur, de sorte à ce qu'il marque. Au final, nous avons eu des résultats qui n'ont pas été optimales puisque le joueur contrôlé marquait moins de but que le joueur Fonceur3Strategy.

A cause de ces problèmes, nous avons décidé de ne pas utiliser l'arbre de décision. Voici les arbres que nous avions obtenus.



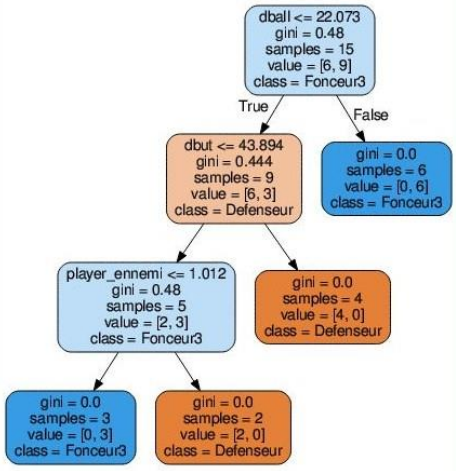


Figure 8 : Exemples d'arbre de décision obtenu à la suite de la simulation

5. Conclusion

Finalement, nous avons décidé d'organiser ce projet de sorte à ce que la classe Stratégie soit écrite le plus simplement possible. Pour cela, nous nous sommes appuyées sur deux classes en particulier, la classe Tools, et la classe Comportement afin de mettre en place 5 stratégies spécifiques. Pour optimiser ces stratégies, nous avons utilisé plusieurs méthodes d'optimisation qui ont plus ou moins aboutis. Ces méthodes ont permis aux stratégies d'être optimales. A travers ce projet, nous avons appris la structure de base d'un codage python de stratégies qui diffèrent en fonction d'un environnement spécifique.