

Grillet Yanis

Piranda Julien

Groupe de TD 7

Rapport de code

Projet Informatique

**Projet informatique : Création d’un baby-foot**

Table des matières

[Introduction 3](#_Toc10794935)

[I) Analyse du problème 3](#_Toc10794936)

[1) Le principe du jeu 3](#_Toc10794937)

[2) Le but du projet 3](#_Toc10794938)

[3) Les hypothèses réductrices adoptées 4](#_Toc10794939)

[II) L’application proprement dite 5](#_Toc10794940)

[1) Les classes utilisées 5](#_Toc10794941)

[2) Les méthodes et classes plus en détail 8](#_Toc10794942)

[a) Les classes du baby-foot 8](#_Toc10794943)

[b) L’initialisation des images de l’IHM 9](#_Toc10794944)

[c) Le fonctionnement du jeu 10](#_Toc10794945)

[III Quelques conclusions 12](#_Toc10794946)

[1) Les tests utilisés 12](#_Toc10794947)

[2) Les limitations 12](#_Toc10794948)

[3) Les apports personnels par rapport au projet initial 12](#_Toc10794949)

[4) Les perspectives et améliorations 13](#_Toc10794950)

[Conclusion 13](#_Toc10794951)

# Introduction

Dans le cadre du projet informatique, nous avons choisi le projet baby-foot. Nous ne savions pas exactement comment aborder le code au début, mais ce sujet nous a interpellé car nous souhaitions réaliser un programme que nous aurions envie d’utiliser même une fois la note rendue et le projet terminé. De plus, ce choix de projet nous permettait de nous concentrer uniquement sur les problèmes liés au code, étant donné que nous connaissions déjà les règles du jeu. Nous avons donc pu réaliser les approximations que nous souhaitions et aborder ce problème avec en tête uniquement les défis du code et le confort de jeu une fois le code finalisé.

# Analyse du problème

## Le principe du jeu

Comme tout le monde le sait, le baby-foot est un jeu se déroulant sur un plateau identique à un terrain de football, avec une balle et des cages. Les joueurs sont fixés sur des barres pouvant coulisser latéralement, et tourner autour de leur axe afin de frapper la balle. Le but est évidemment de marquer des points en faisant rentrer la balle dans les cages de l’équipe adverse.

## Le but du projet

Le but de notre programme était de réaliser un baby-foot virtuel jouable. Le principal était de pouvoir simuler une balle, des joueurs et un terrain de jeu grâce à une interface homme-machine. Les joueurs devaient être déplaçables grâce à l’appui sur des touches du clavier de l’ordinateur, la balle devait rebondir sur les joueurs et les murs, et les buts devaient être détectés et le score affiché sur l’interface. La balle était d'abord lancée à une vitesse V0 au milieu du plateau, avec un angle aléatoire pour qu'une équipe ou l'autre ait la balle avec une chance égale.

Nous avions pensé à diverses manières d’agrémenter notre programme, par exemple avec des images et célébrations pour les buts marqués afin de le rendre plus divertissant., ou encore avec un choix des terrains ou des balles utilisables.

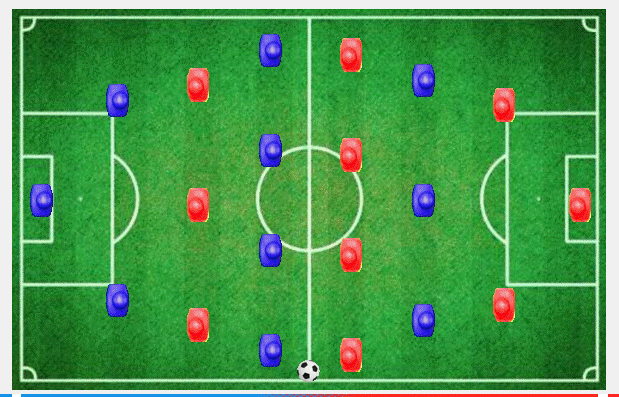
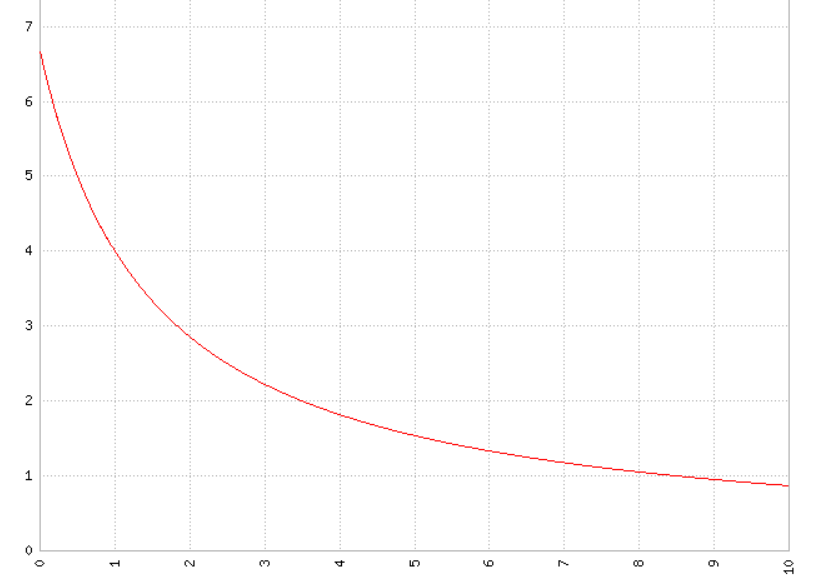


Figure 1 - IHM souhaitée pour notre baby-foot

## Les hypothèses réductrices adoptées

Evidemment, un baby-foot virtuel présente de nombreuses limitations face à un baby-foot classique. Tout d’abord, les contrôles des 4 barres en même temps, selon 2 degrés de libertés (translation et rotation) ne sont pas facilement réalisables pour l’utilisateur de l’interface. Or notre but était de réaliser un jeu divertissant plus que réaliste. Nous avons donc décidé que la rotation ne serait pas possible. Ainsi, le tir dans la balle est automatique. Lorsqu’elle rebondit sur un joueur, sa vitesse est augmentée automatiquement, afin de simuler un tir par rotation de la barre. Ensuite, le contrôle des 4 barres individuellement demanderait une trop grande concentration, c’est pourquoi nous avons décidé que toutes les barres bougeraient de manière solidarisée. Lorsque l’utilisateur appui sur la touche correspondant à *monter,* toutes les barres montent jusqu’à l’arrêt de l’appui, ou la rencontre avec un mur. Ensuite, concernant les frottements et la vitesse de la balle, nous avons préféré réaliser un projet informatique plutôt que physiquement viable. Notre balle a donc une vitesse qui est remise à son maximum à chaque fois qu’elle rebondit sur un joueur, et le reste du temps, elle a une vitesse décroissante en 1/t jusqu’à une vitesse minimale, selon la formule suivante :

Avec un modifiable. (nous avons décidé d’utiliser un de 2, après divers tests pour voir quel permettait d’avoir la vitesse de jeu la plus agréable.



V(t) = + α

Vitesse minimal α

Vitesse

Temps

Figure 2 - Vitesse de la balle en fonction du temps

Enfin, la dernière approximation utilisée concerne les rebonds de la balle. Les rebonds sur les murs respectent les lois de la réflexion de Snell-Descarte :

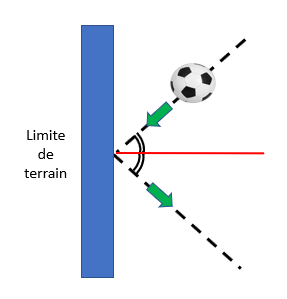


Figure 3 - Réflexion de la balle sur les murs du terrain

Mais lorsqu’elle rencontre un joueur, la balle n’obéit pas à cette loi. En effet, nous avons essayé de jouer comme ceci, et il était difficile de viser les cages correctement avec de tels rebonds. Ainsi, nous avons préféré lui donner la direction des cages adverses avec un angle qui varie aléatoirement lorsqu’elle touche un joueur. Ce système permet de rendre le jeu plus dynamique en facilitant son utilisation. Ainsi en augmentant les risques de buts, la partie sera moins ennuyante.

# L’application proprement dite

## Les classes utilisées

Le code est réparti en 4 fichiers. Un concernant les classes principales utilisées, à savoir les barres, la balle et le plateau notamment, un autre contenant l’affichage des images pour l’IHM, un pour le fonctionnement du programme, et un dernier pour les tests unitaires.

Important : pour lancer le jeu, il faut exécuter le fichier appelé « jeu ».

Dans ces fichiers, l’architecture des classes est réalisée selon les schémas suivants :

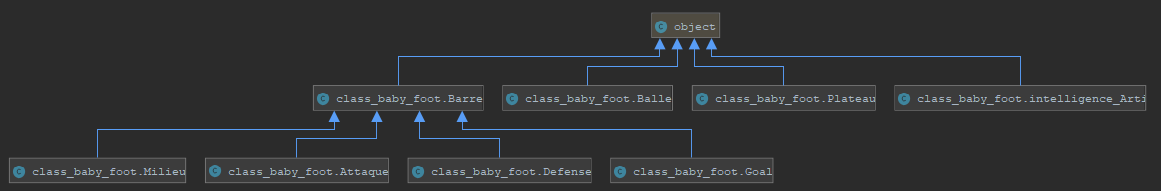


Figure 4 - Diagramme des classes du baby-foot

Les classes héritant de la classe Barre définissent uniquement la position initiale des joueurs en fonction de l’équipe (bleue ou rouge). La classe Balle définit toutes les caractéristiques de la balle, à savoir son rayon, sa position de départ, sa vitesse, sa direction etc… La classe plateau définit uniquement les limites du terrain, ainsi que la position des cages. Enfin, la classe intelligence\_arti permettra le fonctionnement de l’intelligence artificielle au cours d’un match solo ou IA contre IA.

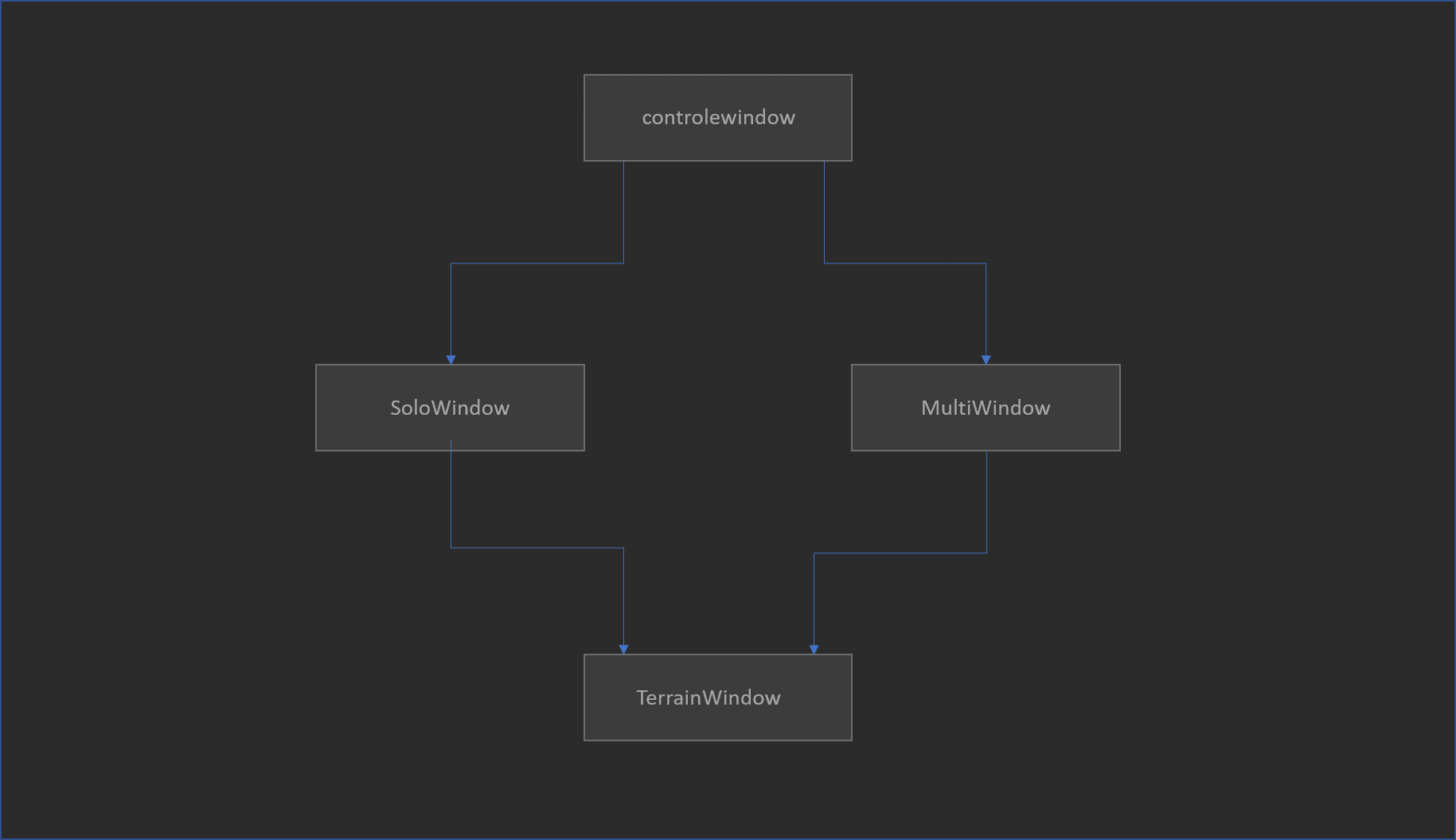


Figure 5 - Diagramme des classes du programme principal

Les classes du programme principale ci-dessus régissent le fonctionnement des différentes fenêtres de l’interface graphique qui vont s’ouvrir. La première est controlewindow. Elle permet le choix de la balle, du terrain et du mode de jeu (nombre de joueurs entre 0et 2)



Figure 4 - ControleWindow permettant le choix des paramètres de jeu

Ensuite, si le choix était multi ou solo, une deuxième fenêtre s’ouvre en fermant la précédente. Cette fenêtre sert juste à expliquer quelles touches utiliser pour le déplacement des joueurs.

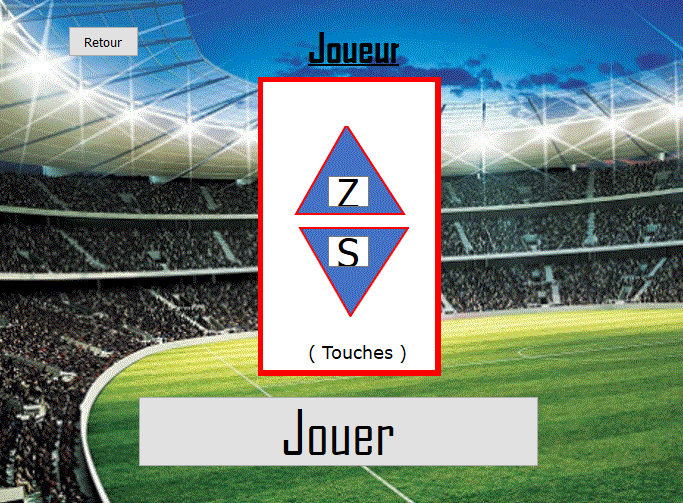
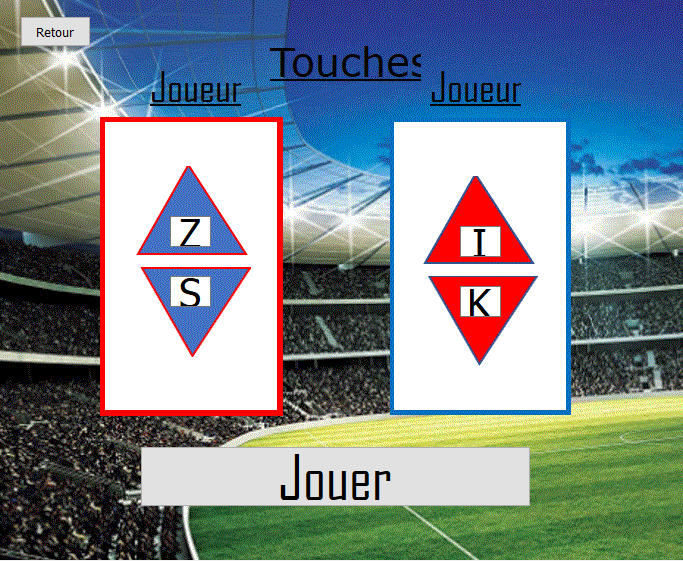
 

Figure 5 - Explication des touches en fonction du mode de jeu sélectionné

Enfin, lors de l’appui sur la touche JOUER, la dernière fenêtre s’ouvre, c’est la fenêtre sur laquelle le baby-foot en tant que tel apparaît, et où le jeu se déroulera. Sur celle-ci, le terrain et la balle choisis sur la première fenêtre s’afficheront, et les joueurs bougeront avec l’appui sur les touches. Il faut appuyer sur START pour lancer le jeu. Un bouton PAUSE permet de mettre le jeu en pause. En mode solo, la difficulté de l’IA peut être modifiée. Enfin, le score s’affiche en direct en haut de la fenêtre, et à chaque but, une image apparaît en bas de la fenêtre, ainsi que le mot «GOAL» sur le côté.

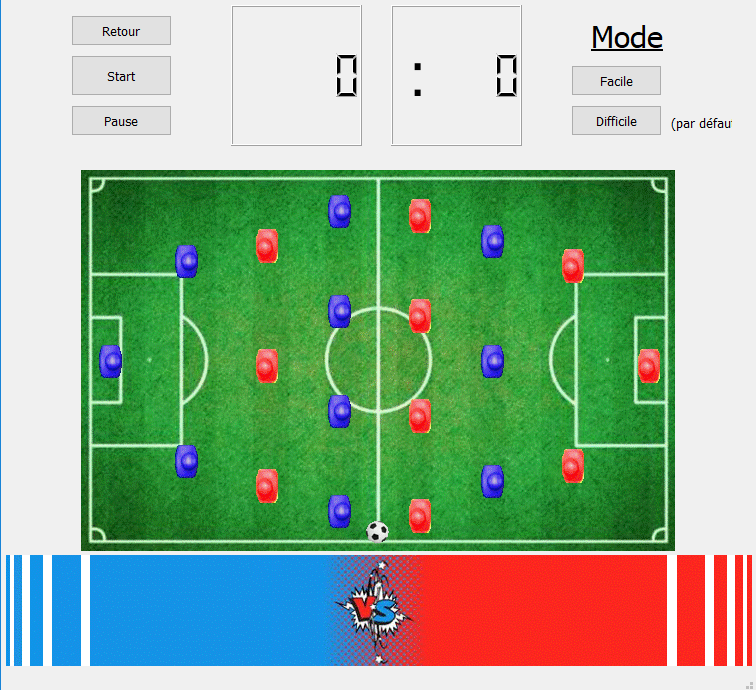


Figure 6 - TerrainWindow permettant de jouer au baby-foot

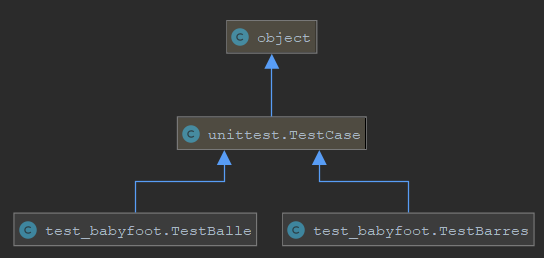


Figure 7 - Diagramme des classes des tests unitaires

L’architecture du fichier des tests unitaires est assez simple. Une classe permet les tests des méthodes associées aux différentes barres, placement et mouvement des joueurs. Une autre permet d’effectuer les tests sur la balle et ses méthodes, donc son placement, sa vitesse et diverses modifications associées.

## Les méthodes et classes plus en détail

Nous avions dans un premier temps réalisé l’entièreté du code dans un seul fichier, à l’exception des tests unitaires, mais dans un soucis de compréhension et de lisibilité du code, nous avons préféré le séparer en trois fichiers plus les tests. Ainsi, la structure des trois fichiers n’est pas foncièrement utile, mais simplement plus agréable.

### Les classes du baby-foot

Dans un premier temps, nous avons créé la partie du code correspondant au fichier des classes du baby-foot. En effet, cette partie est le squelette de notre programme, permettant que le jeu fonctionne sans prendre en compte l’IHM.

Nous avons réalisé une classe Barre, dans laquelle il y a uniquement les variables d’instances correspondant à l’équipe à laquelle la barre appartient, une liste vide pour les joueurs et une autre pour les labels (les images des joueurs pour l’IHM). Il y a ensuite deux méthodes, les méthodes Move\_High et Move\_Low, qui servent évidemment à déplacer les barres en haut ou en bas, en vérifiant si une ou plusieurs des barres touchent un des bords du terain. Ensuite, nous avons créé toutes les classes qui héritent de la classe Barre, à savoir Goal, Defense, Milieu et Attaque. Dans chacune de celles-ci, la liste joueur est initialisée avec les positions des joueurs en fonction de l’équipe.

La classe plateau sert uniquement à définir les limites du terrain et des cages.

La classe Balle est la plus complexe de ce fichier. En effet, elle prend en compte les caractéristiques de la balle telles que sa position, son rayon, sa direction et sa vitesse (grâce à la variable d’instance ‘pas’, mais c’est aussi dans cette classe que sont stockées les informations relatives au nombre de but de chaque équipe, ou encore le temps de jeu depuis le lancement ou le dernier contact avec un joueur, ce qui permet de gérer la vitesse de la balle. Concernant les méthodes associées à la classe Balle, il y a d’abord les deux permettant de contrôler sa vitesse : vitesse\_balle, et sa direction : avancer. Ensuite, il y a les méthodes permettant de gérer les rebonds de la balle. D’abord, la méthode test\_terrain, qui prend en entrées les limites du terrain ainsi que des cages, et la position de la balle . Ici, on test si la balle atteint une des limites du terrain et, le cas échéant, s’il y a but. Si elle touche un bord sans être dans les cages, elle suit les relations de réflexion de Snell-Descartes, et sa vitesse continue de suivre la loi décroissante proposée. Cette méthode renvoie la nouvelle position de la balle. Ensuite, il y a la méthode test\_barre, qui prend en entrée la position de la balle, la liste de joueur d’une barre, la position des cages et un entier qui dépend de la difficulté de jeu. Ainsi, lorsque la balle rentre en contact avec un des joueurs, sa nouvelle direction est vers le centre des cages, avec une variation aléatoire introduite dans le cas ou le dernier paramètre lié à la difficulté n’est pas nul. Cette méthode renvoie aussi la nouvelle position de la balle. Finalement, grâce à la méthode détection\_but, on vérifie si un but a été marqué ou non. Cette méthode prend en entrée uniquement les limites du terrain en x. En effet, dans le cas ou la balle atteint une limite du terrain en x sans rentrer dans les cages, la méthode test\_terrain est effectuée en premier, et le but ne sera pas comptabilisé. Ensuite, si la position de la balle est inférieure à xmin, alors il y a but pour les rouges et si elle est supérieure à xmax alors il y a but pour les bleus.

Enfin, la classe intelligence\_Arti définit juste l’équipe de l’intelligence artificielle (rouge par féfaut car en solo, l’utilisateur incarne l’équipe bleue), un compteur et une liste contenant deux chaînes de caractère : «high» et «low». Ceci permettra de réaliser une fonction aléatoire pour faire bouger l’intelligence artificielle en haut ou en bas.

### L’initialisation des images de l’IHM

Ce fichier ne comporte pas de classes, et permet uniquement la définition des images (labels) à afficher sur le terrain de baby-foot. Pour cela, on définit d’abord une fonction création\_des\_labels, qui permet de créer un fichier .txt, avec toutes les instructions permettant la création des labels associés aux « memes » utilisés à chaque but marqué. Il nous a ensuite suffit de copier-coller l’ensemble du fichier .txt dans la suite du programme, au lieu de le retaper 22 fois.

Ensuite, la fonction définition\_des\_images permet de définir tous les labels associés aux joueurs, aux images 1,2,3 de lancement, au mot « goal » apparaissant à chaque but, au terrain, à la balle etc… Soit à toutes les images dont nous avons besoin pour la partie de baby-foot. Toutes ces images sont crées et ajoutées à la fenêtre. Toutefois, nous n’avons pas besoin de toutes ces images en même temps. C’est pourquoi nous plaçons celles dont nous n’avons pas besoin en continuité (les memes, le 1,2,3 …) en dehors de la fenêtre affichée, puis d’autres fonctions permettent de les déplacer aux coordonnées souhaitées, qui seront appelées en temps voulu. Ainsi, nous avons défini les fonctions affichage\_meme, affichage\_stickman, affichage\_balle et affichage\_joueurs.

La fonction affichage\_balle prend en entrée l’objet Balle, et le label correspondant et permet de placer l’image de la balle à la position de l’objet. De même avec les joueurs, sauf que la fonction le fait pour toute la liste des joueurs d’une barre.

Ensuite, pour l’affichage du Stickman, partie inutile du code pouvant être considérée comme un « easter egg », la fonction permet d’afficher chacune des 7 images du Stickman en boucle, pour donner l’illusion qu’il court, et de les déplacer pour le faire avancer le long de la fenêtre, dès l’appui sur la touche « V » .

Concernant les memes, nous avons aussi rédigé une fonction de sélection des memes, afin qu’ils soient liés au contexte du match. Ainsi, au début ce sont des memes classiques affichés. Lorsque la différence de points est supérieurs à 5, des memes plus « agressifs » sont affichés, et en cas d’égalité, les images apparaissant sont adaptées à cette situation.

Ensuite, la fonction affichage\_meme vient les placer aux coordonnées souhaitées dès qu’elle est appelée, soit à chaque but.

### Le fonctionnement du jeu

Dans le fichier de jeu, nous avons réalisé l’IHM à proprement parler, ainsi que le déroulement d’une partie. Nous commençons par l’affichage de la fenêtre de contrôle, en figure 4. Dans cette classe, nous avons défini 3 types de méthode. Les méthodes ballonX permettent de choisir la balle utilisée en jeu, en donnant une valeur entre 0 et 4 à la variable d’instance balle. Plus tard, cette variable correspondra à l’image choisie pour représenter la balle en jeu. De même, les méthodes radioX permettent le choix du terrain en modifiant la valeur de la variable d’instance fond. Ensuite, les méthodes solo, multi et robots permettent de déterminer le mode de jeu, et d’ouvrir la fenêtre suivante correspondant (La fenêtre solo ou multi, ou bien dans le cas de l’appui sur le bouton robot, la fenêtre de jeu directement). De plus, elles permettent de définir les variables globales balle\_choisie, terrain\_choisie, et AI\_1 et AI\_2. Les variables AI\_1 et 2 permettent de déterminer si les barres des équipes bleues et rouges doivent bouger toutes seules ou non, en fonction du mode de jeu choisi. Enfin, nous avons connecté les boutons de la fenêtre crée aux méthodes définies dans cette classe.

Ensuite, nous avons définis les classes MultiWindow et SoloWindow. Ces fenêtres servent uniquement à expliquer les touches de contrôle aux joueurs. Nous y avons défini deux méthodes, la méthode Jouer, et la méthode Retour, qui permettent respectivement d’ouvrir la fenêtre de jeu, ou la fenêtre de contrôle initiale.

Enfin, la classe TerrainWindow permet d’afficher la fenêtre de jeu, et de le faire tourner. Après avoir déterminé les nombreuses variables d’instance de cette classe, qui correspondent entre autre à la liste des noms de fichiers des terrains et des balles, à l’attribution des barres du fichier baby-foot, à la vitesse initiale de la balle ou à son rayon … Nous appelons la fonction définition des images du fichier set\_up\_pictures, puis sa fonction affichage\_balle.

Ensuite, nous avons défini les méthodes à appeler lors d’un appui sur un bouton de la fenêtre. A savoir, le bouton start, le bouton pause, retour, facile ou difficile. Le bouton Start lance la partie et le mouvement automatique de la balle grâce à la méthode simulation, qui lance un timer. Ainsi, après un certain temps, cette méthode en appelle une autre, appelée confirmation\_prochain\_tour.

Cette dernière permet l’affichage du 1,2,3 qui lance la partie, mais surtout le lancement de la prochaine itération du programme qui redéfinit toutes les coordonnées des images et objets. Elle permet aussi de gérer le cas d’un but et de remettre la balle au centre et d’afficher le meme correspondant le cas échéant. Enfin, cette méthode appelle la méthode Prochain\_Tour.

Celle-ci affiche à chaque fois les images des joueurs et du stickman grâce à l’appelle de la fonction correspondant dans le fichier set\_up\_pictures.. Ensuite, elle fait fonctionner les intelligences artificielles si nécessaire, puis appelle la méthode avancer\_la\_balle.

Avancer\_la\_balle permet d’effectuer tous les tests définis dans le fichier class\_baby\_foot, puis de faire avancer la balle en fonction de sa vitesse et de sa direction après les tests. Dans le cas d’un but, on appelle la méthode compteur de points.

Cette méthode sert à mettre à jour l’affichage des points, et à déterminer dans quel configuration on se situe pour l’affichage des memes (quelle équipe vient de marquer, et quelle est la différence de points).

Finalement, nous avons défini une méthode move\_the\_rods. Celle-ci permet d’appeler les méthodes move\_high et move\_low du fichier class\_baby\_foot, et donc de faire bouger les joueurs d’une équipe vers le haut ou vers le bas. Cette méthode est appelée par la méthode keypressevent. Cette dernière permet de détecter l’appui sur une des touches paramétrées (dans notre cas Z, S, I, K et V), et d’appeler la méthode qui nous convient (dans notee cas, la méthode move\_the\_rods) lors d’un appui.

Pour simplifier tout ceci, on peut dire que cette classe et ses méthodes respectent le graphique suivant :

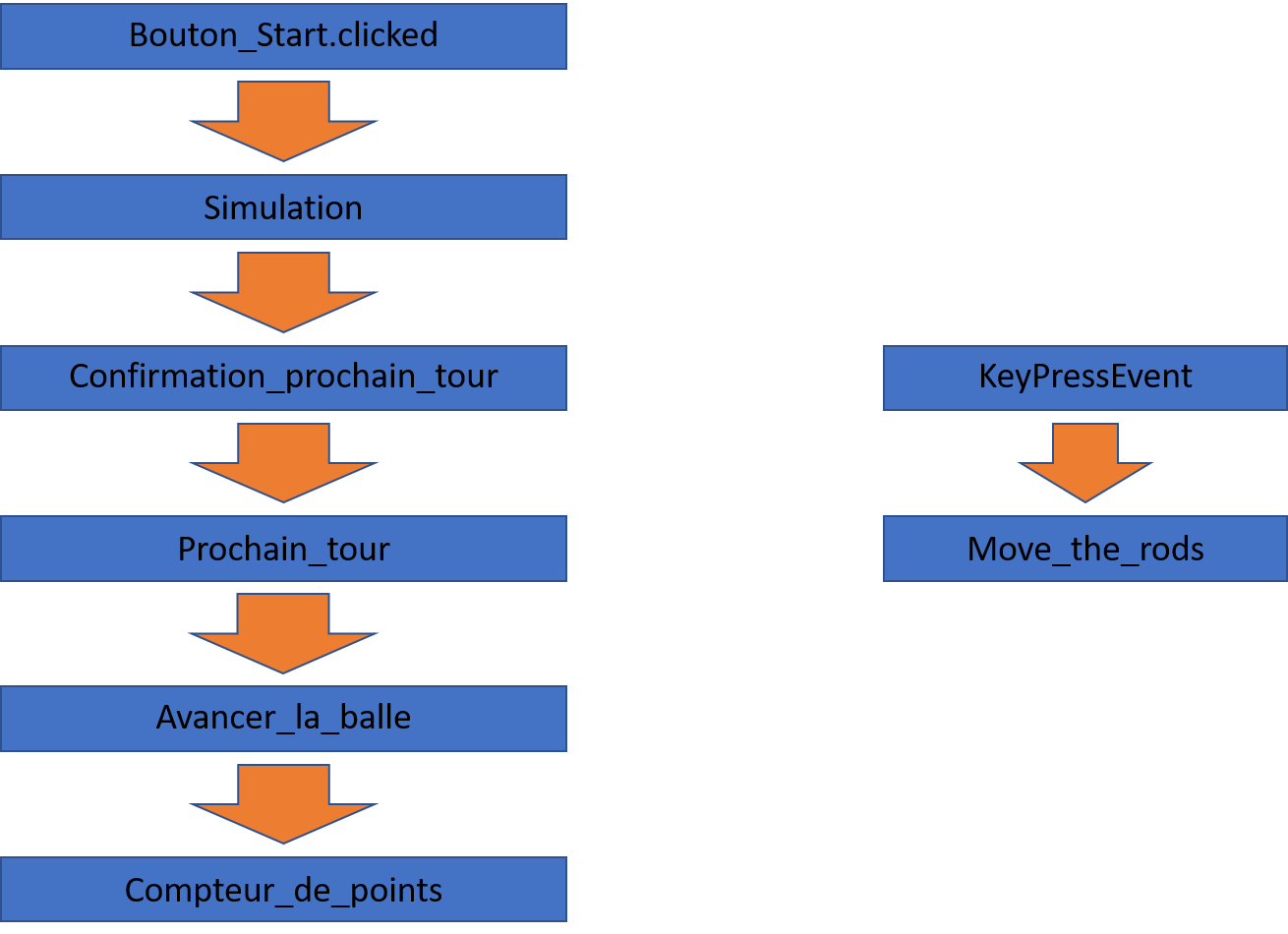


Figure 8 - fonctionnement de la classe terrain\_window

# III Quelques conclusions

## Les tests utilisés

Afin de tester le programme, nous avons créé un quatrième et dernier fichier, celui des tests unitaires. Nous avons vérifié notamment la bonne position des joueurs à l’initialisation des classes héritant de la classe Barre, puis les méthodes permettant de bouger les barres, c’est-à-dire est-ce que la méthode fait bien bouger les barres, et est-ce que si une barre touche une limite de terrain, elle bouge ou non.

Ensuite, nous avons testé la classe Balle, avec des tests sur ses variables d’instance, puis sur ses méthodes. Notamment un test sur sa méthode vitesse, et un sur sa méthode test\_barres. Nous avons vérifié que la vitesse et direction étaient bien modifiées comme prévu en cas de contact avec un joueur, mais aussi sans contact.

## Les limitations

Les limitations rencontrées sont nombreuses. La plus gênante est le fait que l’appui sur deux touches en simultané ne fonctionne pas. En effet, avec la méthode KeyPressEvent, nous pouvons détecter l’appui sur une touche ou une autre, mais pas sur deux en même temps. Le jeu en multijoueur est donc très limité par ceci, puisqu’un appui en continu ne fonctionne pas et qu’il faut appuyer rapidement sur les touches pour pouvoir jouer.

Ensuite, les 4 barres d’une équipe bougent en même temps, ce qui n’est pas représentatif de la réalité. Ceci n’est pas une limitation liée au code, mais au côté pratique du jeu. En effet, il paraît difficile de se concentrer sur les barres en même temps, et de ne pas se tromper de touche pour bouger une barre, puisqu’il faudrait 8 touches pour gérer les 4 barres indépendamment. De plus, si l’appui en simultané sur plusieurs touches n’est pas possible, cette manière de procéder paraît encore plus complexe.

Par ailleurs, il y a un temps de latence entre l’appui sur une touche et le déplacement du joueur, lié directement au fonctionnement de KeyPressEvent, et qui ne peut donc pas être amélioré avec ce procédé.

## Les apports personnels par rapport au projet initial

Comme nous l’avons précisé plus tôt, nous avons apporté de nombreuses modifications au sujet initial. Premièrement, dès que nous avions réalisé l’IHM, nous avons senti qu’il nous restait du temps et avons eu envie d’améliorer les possibilités de jeu. C’est pourquoi nous avons rajouté le choix de la balle et du terrain, l’affichage des memes et le Stickman.

## Les perspectives et améliorations

Nous nous rendons compte évidemment que le code est largement améliorable. En premier lieu, nous pourrions certainement trouver un moyen de le rendre plus réactif à l’appui sur une touche, afin d’éviter le temps de latence entre l’appui et le déplacement des joueurs. Ensuite, il existe certainement une astuce pour détecter l’appui sur plusieurs touches en simultané, ce qui permettrait de jouer à plusieurs sans gêne, ou bien de contrôler chaque barre individuellement.

# Conclusion

Le bilan de ce projet informatique aura été très positif pour notre binôme. En effet, avoir fini un projet qui paraissait difficilement réalisable avec nos connaissances en Python en milieu d’année, voire infaisable en début d’année, est très réjouissant. Par ailleurs, nous avons pu mettre en œuvre les connaissances que nous avons acquis de manière théorique, et les mettre en pratique a été très formateur pour nous. A notre sens, ce type de projets est la meilleure des manières de progresser en programmation. Cela nous a permis de découvrir de nombreuses manières de résoudre les problèmes auxquels nous avons été confrontés, et nous avons pu acquérir le réflexe d’aller chercher sur internet lorsque nous ne connaissions pas le module ou l’instruction nécessaire. Enfin, être libre sur le choix du sujet et sur le rendu final nous a laissé une grande marge de manœuvre, ce qui nous a permis de prendre du plaisir à réaliser des fonctions supplémentaires à notre programme, plutôt que de se contenter du strict minimum attendu. Ainsi, nous avons maintenant un programme que nous pourrons réutiliser à souhait, et l’envie, pourquoi pas, de créer d’autres programmes.