Projet Java-Graphique-IHM

Soccer-Stat

Projet réalisé par :

Sacha LENARTOWICZ
Raïhane SIDQUI

Polytech Paris-Sud 2015-2016

1ère année du cycle d'ingénieur, spécialité informatique

SOMMAIRE

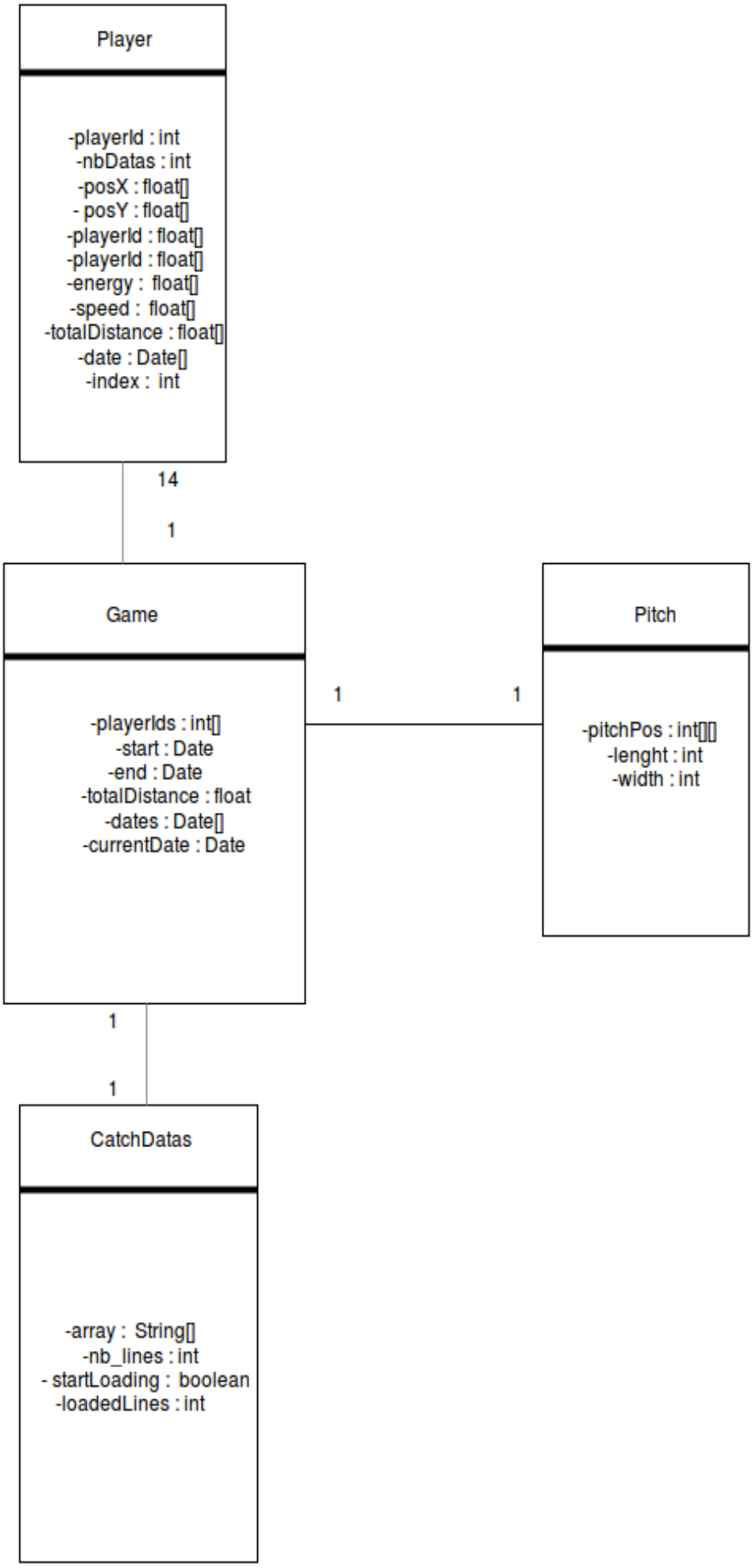
I- Le sujet

Le but principale du sujet était de développer une interface capable d'afficher sur un terrain des joueurs de football, modélisés par des flèches, les données des joueurs à affichés étant fournies. Différentes fonctionnalités devaient être présentes dont les suivantes :

- sélectionner les joueurs à afficher.
- placer la caméra virtuelle à la place de la tête d'un joueur.
- modifier la vitesse de jeu.
- afficher une ligne représentant la trajectoire des joueurs.
- afficher une carte thermique représentant le positionnement des joueurs sur le terrain.

Afin de réaliser la partie applicative du projet, il était demandé de faire un diagramme UML. Voici le diagramme UML qu'on a pu mettre à jour au fur et à mesure de l'avancement de notre projet :

II- Diagramme UML


Player
-playerId : int
-nbDatas : int
-posX : float[]
- posY : float[]
-playerId : float[]
-playerId : float[]
-energy : float[]
-speed : float[]
-totalDistance : float[]
-date : Date[]
-index : int
14
1
Game
-playerIds : int[]
-start : Date
-end : Date
-totalDistance : float
-dates : Date[]
-currentDate : Date
1
1
Pitch
-pitchPos : int[][]
-lenght : int
-width : int
1
1
CatchDatas
-array : String[]
-nb_lines : int
- startLoading : boolean
-loadedLines : int

III- Lecture et chargement des données

Après avoir effectué le tutoriel proposé lors de la première séance de TP, nous avons commencé à nous intéresser à la capture des données contenues dans les fichiers des mi-temps. Nous nous sommes renseigné sur le contenu des données, et nous avons découvert, grâce au site proposant de téléchargé les données, qu'une ligne correspondait à un enregistrement pour un seul joueur, à un moment donné du match.

Les données étant enregistrées tous les 5 centièmes de secondes, leur nombre était important et nous nous sommes demandés si le fait de les charger d'un seul coup au début de l'exécution ne poserait pas un problème. Notre professeur de TP nous a confirmé que l'utilisation d'un buffer n'était pas nécessaire, et il avait effectivement raison (sur nos machines, le chargement prend un peu moins de 10 secondes pour une mi-temps entière).

Nous avons donc commencé le chargement. Nous avons décidé de stocker chaque enregistrement dans des objets de classe Player. Nous aurions pu choisir qu'un enregistrement concernait l'ensemble des données d'une date précise, ce qui nous aurait évité d'ajouter aux Player un champ « index », indiquant l'indice auquel on se trouve lors du déroulement de la partie.

C'est l'objet de classe CatchDatas qui est chargé d'ouvrir et lire le fichier de données. Un objet de classe Game est créé, créant à son tour la totalité des joueurs présents dans la période chargée. Pour chaque joueur, nous avons choisi de stocker les données dans des tableaux plutôt que dans des ArrayList. Bien que les tableaux soient de taille statique (nous obligeants, pour les joueurs faisant leur entrée sur le terrain tardivement à allouer de la place en mémoire inutilement), nous pensions qu'il était possible que l'utilisation des ArrayList aurait ralenti l'exécution du programme.

Nous avons également pris soin d'instancier un objet de classe Pitch (terrain), contenant un tableau d'entier indiquant le nombre de fois qu'un joueur a été présent à un certain mètre carré. Nous nous sommes rendu compte, à la fin du projet, qu'il fallait instancier ce tableau pour chaque joueur, mais nous n'avons de toute façon pas eu le temps de développer l'affichage de cartes graphiques.

Il nous était demandé de vérifier le chargement des données à l'aide d'une classe SimpleTest, dont nous ne connaissions pas le fonctionnement. Nous avons effectué les deux premiers tests

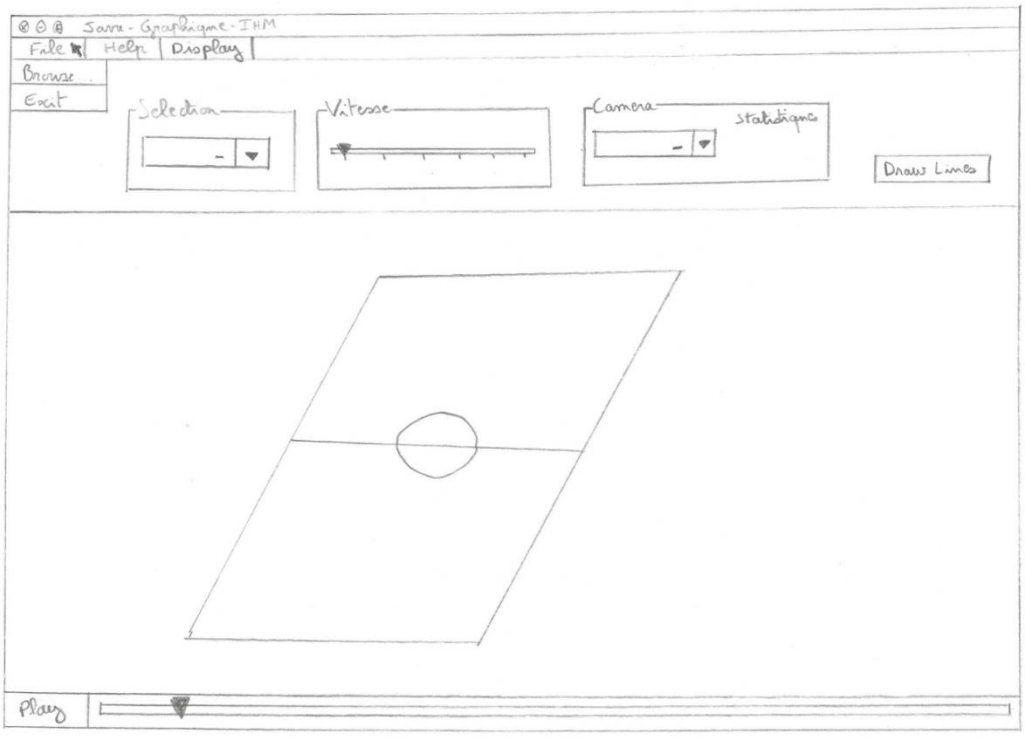
demandés en plus de tests que nous avions programmés pendant l'écriture du code. Cette partie était très intéressante et cet outil continuera certainement à nous servir dans le futur.

IV- Création de l'interface

La seconde tâche sur laquelle nous nous sommes penchés était la création d'une interface graphique. Nous avons tout d'abord repris le code du tutoriel de la première séance de TP. Nous avons ensuite cherché à afficher pour chacun des joueurs un cube, devant se déplacer en fonction de la position du joueur. Nous avons tout simplement réutilisé la méthode SimpleUpdate, appelée automatiquement par l'ordinateur.

Après avoir réussi cet affichage, nous avons constaté que les joueurs se déplaçaient un peu trop vite. La méthode SimpleUpdate est en fait appelée, sur nos machines, tous les 2 centièmes de seconde environ. Or, l'intervalle de temps entre deux données d'un même joueur est de 5 centièmes de seconde. Il a donc fallu exécuter ce qui se trouve dans la méthode SimpleUpdate uniquement lorsque cela était nécessaire. Ainsi, une seconde dans la réalité correspond à une seconde dans l'application.

Avant de continuer, nous avons réalisé un croquis représentant notre interface. Il fallait que l'interface puisse présenter les différentes fonctionnalités de façon claire.
Le croquis que nous avions réalisé est le suivant :



Vous pouvez constater, avec la capture d'écran suivante, que nous avons été fidèles à ce que nous avions projeté :

Pour l'affichage des joueurs, il a fallu résoudre un premier problème, résumé par la question suivante : comment sélectionner les données des joueurs devant être affichés ? Nous ne pouvions pas simplement incrémenter un indice commun à tous les joueurs, car les joueurs n'apparaissant qu'à la fin de la période auraient été présents dès le début. De plus, des enregistrements sont parfois absents dans les données, ce qui aurait fini par créer des décalages. La solution était, comme nous l'avons expliqué auparavant, d'incrémenter un index pour chaque joueur, et de vérifier avant de l'incrémenter si la date de l'enregistrement du joueur correspond bien à la date actuelle, stockée dans l'objet Game.

Nous avons finalement affiché les flèches que vous nous avez fournies avec le terrain à la place des cubes. Pour la rotation des joueurs, il était indispensable de pouvoir l'effectuer par rapport au repère associé au terrain. La fonction rotate() nous a posé problème, car elle effectuait un rotation par rapport à la rotation précédemment effectuée (du coup, les joueurs tournaient en rond sans cesse). La solution était d'utilisé la fonction setLocalRotation(), appliquée sur la forme

géométrique et prenant en paramètre un Quaternion. Enfin, nous pouvions commencer à ajouter les diverses fonctions demandées.

V- Les fonctionnalités

A- La vitesse de lecture et la navigation

Pour pouvoir sélectionner la vitesse de lecture, il nous semblait approprié d'utiliser un JSilder. Il est possible de sélectionner une vitesse entre 1x et 10x. Nous avons pour cela récupéré la valeur du JSlider, et nous avons sélectionné la date à afficher en fonction de cette valeur dans SimpleUpdate. Pour des vitesses de lecture élevées, certains enregistrements ne sont donc « sautés » et ainsi pas affichés.

Par la même occasion, nous avons créé un JSlider permettant à l'utilisateur de naviguer dans l'animation du match (comme il est possible de naviguer dans une vidéo). Il était facile de pouvoir arrêter la lecture, en empêchant d'effectuer les instructions dans simpleUpdate(). Nous avons donc placé à gauche du JSilder un JButton pour pouvoir mettre pause lors de la lecture, et redémarrer la lecture lorsque l'application est en pause. Le texte du bouton est modifié, suivant l'état de l'application.

B- La sélection des joueurs

Une des fonctionnalités qu'on devait implémenter consistait à permettre à l'utilisateur de sélectionner les joueurs qui sont affichés sur le terrain. Pour ce faire, nous avons décidé d'utiliser une JComboBox qui affichera tout simplement les numéros des joueurs présents sur le terrain ainsi que deux autres options « Tous les joueurs » et « Aucun joueur ».

L'idée était de faire disparaître des joueurs et les faire réapparaitre dans notre interface 3D. C'était une question d'invisibilité. Et donc, l'astuce était d'utiliser la méthode

setCullHint(CullHint.Always) sur le joueur en question (dans notre cas, le joueur est un Spatial) pour le rendre invisible. Même principe pour le faire réapparaitre avec CullHint.Never.

À la fin, notre fonctionnalité marchait sans problème, mais on s'est rendu compte qu'on aurait pu l'optimiser et utiliser une JCheckBox à la place pour pouvoir cocher ou décocher un joueur.

C- La sélection de la vue

La sélection de la vue avait un niveau de priorité 3 et on a réussi à l'implémenter non sans difficulté. Il fallait pouvoir placer la caméra virtuelle sur un joueur donné afin d'offrir à l'utilisateur la vision de ce joueur et ensuite de pouvoir revenir à la caméra par défaut.

En soi, ce n'était pas compliqué à faire. On a crée une caméra de type CameraNode qu'on attachait sur le nœud qui contenait et le joueur (Spatial) et son numéro (Bitmap) avec attachChild ensuite on alterne entre la caméra et ChaseCamera en utilisant la méthode setEnabled(boolean) .

Cette fonctionnalité a pris un peu plus de temps que prévu car on essayait au début de détacher la caméra du nœud du joueur pour pouvoir revenir à ChaseCamera et ce n'était pas le bon raisonnement à suivre. Quant au composant Swing, on a choisi une JComboBox de la même manière que pour la sélection des joueurs.

D- Affichage du numéro des joueurs

L'affichage des joueurs consistait à utiliser des BitmapText et modifier le texte de ces derniers afin qu'ils prennent le numéro des joueurs. On a bien entendu dû faire une conversion d'entier en chaîne de caractère. Et ensuite, on attachait le BitmapText dans le nœud qui contient le joueur correspondant (Spatial).

La petite difficulté résidait à faire en sorte que le BitmapText soit toujours visible quel que soit la position de la caméra. Pour ce faire on a utilisé un BillBoardControl et qu'on fixé son Alignement à l'alignement de l'écran avec setAlignment(BillboardControl.Alignment.Screen) ainsi peu importe la rotation de l'écran le BitmapText fera toujours face à l'écran. Enfin on ajouté le BillBoardControl au BitmapText avec la fontion addControl.

E- La trace des joueurs

La dernière tâche dont nous nous sommes occupés a été l'affichage de la trace des joueurs. Pour chaque joueur, nous avons décidé d'afficher une trace, composée de 80 courtes lignes, ce qui correspond à 4 secondes de jeu. Nous devions récupérer les 80 dernières positions pour chaque joueur, afin de connaître les points entre lesquels tracer ces 80 lignes.

Il a été nécessaire d'instancier une matrice de Mesh (1ère dimension pour les joueurs, deuxième dimension pour les 80 lignes) ainsi qu'une matrice de Geometry de même dimension. Lors de l'affichage des lignes, on regarde la vitesse du joueur. Si à cet instant, il marche au ralenti, alors la ligne sera bleue. Pour une vitesse plus élevée, la ligne sera jaune et pour un sprint, elle sera rouge.

Pour activer l'affichage, nous avons tout simplement ajouté un JButton, permettant d'afficher et de supprimer les traces et dont le texte change suivant l'affichage ou non des trajectoires. Lors du chargement d'un nouveau fichier, les lignes s'effacent et le traçage est désactivé par défaut.

F- La barre de menu

Pour faire notre barre de menu, on a utilisé trois composants Swing : JMenuBar pour la barre de menu, ensuite JMenu pour les différents menu qu'on propose. Nous en avons deux :

- File qui contient deux JMenuItem :
  - Browse : pour pouvoir charger un fichier auquel on ajoute un Listener où on crée notre JFileChooser.
  - Exit : pour pouvoir quitter notre application et donc notre principale JFrame.

- Help qui contient un JMenuItem :
  - Get controls : qui crée une nouvelle fenêtre qui contient un JPanel qui contient JTextArea où on explique le fonctionnement et comment on utilise notre application. Et enfin un bouton 'ok' qui permet de fermer la fenêtre.

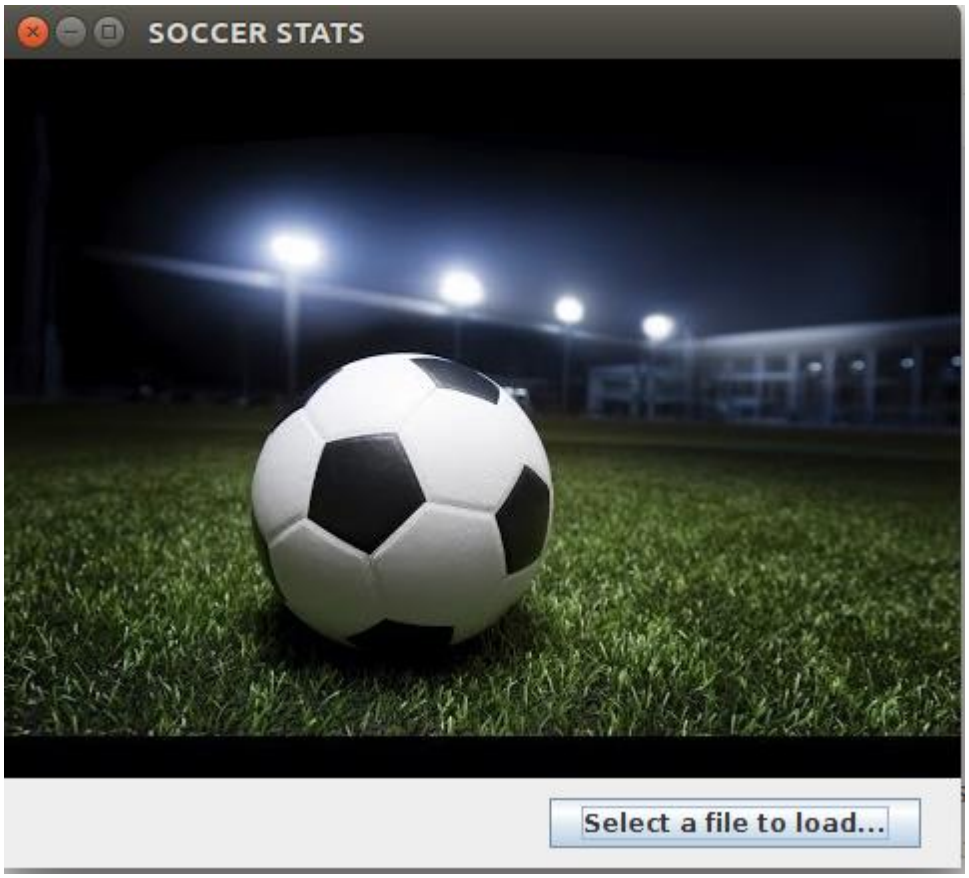
G- La fenêtre d'accueil

Notre petite touche originale. En effet, au départ on a voulu afficher une fenêtre de chargement (Loading) contenant une jolie image d'un terrain de foot et d'un ballon afin de divertir l'utilisateur pendant le chargement du fichier.

Cette fenêtre contenait à la base une JProgressBar qui dure plusieurs secondes, le temps que l'utilisateur appuie sur un bouton Browse et choisit un fichier à charger. Cependant, nous n'avons pas réussi à envoyer la bonne donnée à la JProgressBar, rendant l'affichage de la progression impossible.

En fin de compte, on a trouvé un bon compromis. En effet, on a décidé de supprimer la JProgressBar et laisser simplement un bouton "Select a file to load" avec l'image sympathique au centre de la fenêtre.

Voici un aperçu de notre fenêtre d'accueil :



VI- Organisation du travail

Au début du projet, nous avons réalisé chacun de notre côté le tutoriel pour apprendre à utiliser JMonkeyEngine. Ensuite, pour la quasi-totalité du projet, nous avons travaillé ensemble. Nous avions envisagé la possibilité d'utiliser un outil tel GitHub, mais nous nous sommes arrangés pour se retrouver après les cours pour avancer ensemble. Ainsi, nous nous sommes assurés du bon fonctionnement d'une fonctionnalité avant de passer à la suivante.

VII- Perspectives d'évolution

Nous aurions aimé avoir le temps d'ajouter à notre programme l'affichage de cartes thermiques, représentant les zones dans lesquelles les joueurs ont été le plus présent. Malheureusement, nous avions un autre projet en cours et nous devions aussi réviser des examens. Toutefois, nous sommes fiers du travail que nous avons réalisé et de la bonne entente de notre binôme.

Nous ne fournissons pas à l'utilisateur de feedback. Les temps de chargement sont tout de même assez long (10 secondes pour charger le premier fichier, un peu plus lorsque l'application simule déjà un période) et il aurait été intéressant de soit modifier le curseur de la souris, soit afficher une barre de chargement, ou encore de griser la partie « fonctionnalité » de l'interface.

La sélection des joueurs à afficher à l'écran se fait à l'aide d'une JComboBox. Pour pouvoir sélectionner un seul joueur, il faut d'abord choisir de n'afficher aucun joueur et de sélectionner le joueur désiré. Autre problème que nous pose la JcomboBox : si on sélectionne un joueur que l'on ne veut pas, il n'est pas possible de revenir en arrière, et il faut effacer tous les joueurs avant de re-sélectionner les joueurs désirés. Il aurait finalement été préférable de créer une liste de case à cocher.