Avant-projet du projet informatique

Matthieu Denoux - Groupe 1

27 décembre 2013

Chapitre 1

Présentation du sujet

1.1 Principe général

Le sujet choisi a pour intitulé **Babyfoot en réseau**. Il s'agit de concevoir un système complet de jeu en réseau. Le système serait donc séparé en deux parties, une serveur et une client. Chaque joueur pourrait donc se connecter à une partie n'ayant pas encore commencé et une fois le nombre de joueurs réunis (*2 ou 4*), la partie serait lancée. Il faut donc réaliser à la fois le système réseau, l'interface graphique et imaginer un gameplay qui rende le jeu agréable.

1.2 L'interface graphique

Au niveau de l'interface graphique, il faudra réaliser plusieurs fenêtres successives de menus pour parvenir jusqu'au jeu lui-même. Ces fenêtres seront réalisées avec la bibliothèque inclue dans le package standard **Swing**. J'ai aussi choisi d'utiliser des JFrame et donc de programmer une application à part entière et non une appliquette.

Menu principal Un premier menu, dit *menu principal*, permettra de commencer une nouvelle partie en ligne, de rejoindre une partie déjà en cours, de modifier les options ou bien de quitter le jeu. Un header sensé unifier les différentes fenêtres sera aussi présent, il donnera en plus une certaine identité graphique au jeu.

Commencer une partie Si l'on commence une nouvelle partie en ligne, on se retrouve dans une « salle d'attente ». Là, il est possible de configurer la partie que l'on souhaite lancer, l'ouvrir à d'autres joueurs puis d'attendre que d'autres joueurs rejoignent la partie. Une fois que les équipes sont complètes, on peut lancer le jeu.

Rejoindre une partie On peut aussi sélectionner une partie dans la liste des parties déjà commencées, dans la limite des places disponibles. On rejoint alors un salon similaire à celui décrit dans la section précédente où l'on attend que la partie soit complète avant que le meneur, celui qui a créé la partie, ne lance le jeu.

Déroulement d'une partie Il y aurait deux gameplays différents :

— l'un entièrement manuel verrait le joueur maître de ses possibilités. Ainsi, les touches « A », « Z », « E », « R »permettraient de sélectionner la canne que l'on souhaiterait manier. Les flèches « Haut »et « Bas »permettraient quant à elle de déplacer les cannes

tandis que la barre espace commanderait le tir. Je ne pense pas réaliser plusieurs puissances de tir possibles mais cela pourrait consister en une amélioration de la richesse du gameplay.

— l'autre, principalement rencontrée dans les autres jeux de babyfoot trouvés en ligne, serait plus automatisée : les cannes se déplaceraient toutes ensemble de manière synchronisée avec les flèches « Haut »et « Bas ». Le tir pourrait être ou non automatisé selon que j'aurais le temps pour configurer cette fonctionnalité supplémentaire.

Options Un petit menu sera consacré aux options, notamment le type de jeu que l'on souhaite utiliser (manuel ou automatisé, auquel cas jusqu'à quel point) et d'autres options.

Chapitre 2

Analyse de la solution envisagée

2.1 Découpage en modules

Je prévois de découper le code en trois modules principaux :
- L'interface graphique (rangée dans /gui)
- La partie réseau de l'application (rangée dans /network)
- Le cœur algorithmique de l'application (rangée dans /core)

Chaque partie comportera donc plusieurs classes qui se chargeront chacune d'une des tâches du module parent.

2.2 L'interface graphique

MainFrame.java Gère la fenêtre qui englobe tout le reste. On utilisera en fait des JPanel pour modifier le contenu de cette fenêtre. J'empêcherai dans un premier temps de modifier la taille de la fenêtre pour éviter d'avoir des problèmes de dessin du terrain de babyfoot à gérer. Cette classe contient le **main** du programme.

BPanel.java Classe abstraite présentant certaines actions mécaniques quant aux caractéristiques des JPanel utilisés (header, background color, taille, layoutmanager, etc.). La plupart des panels ci-dessous en héritent.

ConnectPanel.java Gère le premier écran affiché à l'ouverture du jeu. On y choisit le pseudonyme que l'on souhaite utiliser dans le reste du jeu et qui s'affichera ensuite sur les écrans lors du contact avec d'autres joueurs.

MenuPanel.java Gère le menu principal affiché après connexion. Contient les boutons qui mèneront vers les principales actions citées plus haut (nouvelle partie, rejoindre, options, quitter).

NewPanel.java Lorsque l'on lance une nouvelle partie, on obtient cet écran qui contiendra les principales options nécessaires pour configurer une partie puis l'ouvrir à des joueurs extérieurs. Une fois cette partie ouverte, on aboutit à une *salle d'attente*.

ServersPanel.java Gère la liste des parties actuellement en recherche de joueurs lorsque l'on cherche à rejoindre une partie. On sélectionne une partie dans cette liste puis on aboutit à une *salle d'attente*.

WaitingRoomPanel.java Une fois la partie configurée et ouverte, on aboutit dans cette salle d'attente qui attendra que certaines conditions soient réunies pour permettre au jeu d'être commencé. On peut *aussi* y accéder depuis la partie *rejoindre une partie*.

GamePanel.java La partie à proprement parler. Donc le conteneur du dessin du terrain qui gère les éléments extérieurs, les événements et toute autre interaction avec le reste du code. Inclut une zone de dessin.

GameZone.java Zone de dessin chargée de représenter le terrain, les joueurs, etc.

ChatPanel.java Une partie un peu à part qui est chargée de gérer l'affichage du chat. Celui-ci sera présent à de nombreux endroits : la liste des parties disponibles, la salle d'attente, la création d'une partie et durant le jeu lui-même. Or, tous ces affichages sont centralisés dans cet unique fichier qui centralise ainsi le traitement.

2.3 La partie réseau

Server.java Gère les différentes requêtes et les redirige vers les autres entités du serveur (chat, jeu, etc.). Contient des thread pour pouvoir gérer simultanément ces différentes actions.

PlayerServer.java Gère la connexion au serveur d'un des joueurs. L'enregistre dans une liste de joueurs connectés. Peut aussi renvoyer la liste des joueurs connectés actuellement et leur état.

ChatServer.java S'occupe de gérer la partie serveur du tchat : écoute un port, récupère les messages envoyés par des joueurs ainsi que le salon où ils se trouvaient à ce moment là. Les enregistre dans un fichier ou bien dans une base de données (à voir).

ChatClient.java S'occupe de gérer la partie client du tchat : une fois instanciée par le ChatPanel, elle permet d'envoyer des données au serveur et de récupérer la liste des messages disponibles dans le salon. Elle permet aussi de changer de salon, d'afficher la liste des salons et celle des joueurs connectés sur le jeu.

2.4 Le cœur algorithmique

Player.java Le bloc de base du cœur algorithmique est le joueur.

Game.java Contient les données principales pour une partie, notamment l'état des scores, les joueurs y participant, l'avancement de la partie, etc.

Database.java Gère la base de données PostGreSQL ou bien textuelle qui pourrait être utilisée par le programme pour stocker des informations. Sera principalement utilisée par le module network.*.

Chapitre 3

Échéancier

1. **Fin Décembre** : rédiger l'avant-projet.
2. **Début/Mi Janvier** : obtenir la validation et les annotations sur la structure choisie.
3. **Fin Janvier** : réaliser la partie graphique du programme et avoir regardé les grandes lignes du développement serveur / bases de données. Avoir mis au point les éléments de base du gameplay (interaction joueur/machine).
4. **Fin Février** : développer le serveur et la gestion des différents types de requêtes. Mettre au point le tchat et la gestion des joueurs avec la base de données.
5. **Mi Mars** : dresser les liens entre serveur et jeu. Tester.