Skyjo Client-Server

Description

Notre projet sera un jeu de cartes avec une implémentation client/serveur.

Le jeu de cartes se prénomme Skyjo (projet de l'année passée).

Les règles du jeu

Le but étant d'obtenir le moins de points possibles en inversant et échangeant des cartes et se joue avec au minimum 2 joueurs et plusieurs spectateurs.

Nous utiliserons donc un système de gestion de base de données (JDBC) pour pouvoir stocker les informations des joueurs durant la partie ,les manipuler et les gérer dans la base de données ainsi que des interfaces graphiques avec FXML pour permettre au client de se connecter,de le mettre en attendre et afficher une erreur.

Notes d'utilisation

Pour qu'un client se déconnecte il ne faut pas tout simplement fermer la fenêtre du client en question mais il faut également arrêter le processus sur netbeans (carré rouge sur netbeans)

Description des classes

Notre projet est subdivisée en plusieurs projet :

- Skyjo : constituant la partie model de notre jeu.
- SkyjoClient : constituant tout ce qui se rapporte à un client.
- SkyjoMessages : constituant tout les messages pouvant être reçu ou envoyés.
- SkyjoServer : constituant tout ce qui se rapporte au serveur.
- SkyjoParent : qui reprend les 4 projets ci-dessus.

Skyjo :

Ce projet comporte les classes suivantes :

- ModelException : classe appellée qui renvoie une exception au cas ou le modèle a échoué.
- Card : représentant les cartes du jeu (càd des cartes ayant comme valeurs de -2 à 12)
- Deck : représentant le paquet de cartes ayant pour chaque valeur de cartes un nombre de cartes associées
- Discard : représentant la défausse sur laquelle les cartes sont déposées ou piocher
- Model : représentant le modèle de notre MVC
- Player : représentant un joueur. Il débute la partie avec 12 cartes
- Status (énumération) : représentant les statuts que le modèle peut avoir ainsi que les messages que le modèle peut utiliser pour notifier la barre d'état
- Randomizer : Permet de générer des indices aléatoires pour 12 cartes d'un joueur au début du jeu

ainsi qu'une interface nommée Facade : qui définira toutes les méthodes que le modèle(Model.java)devra mettre en oeuvre.

SkyjoClient :

Ce projet comporte les classes suivantes :

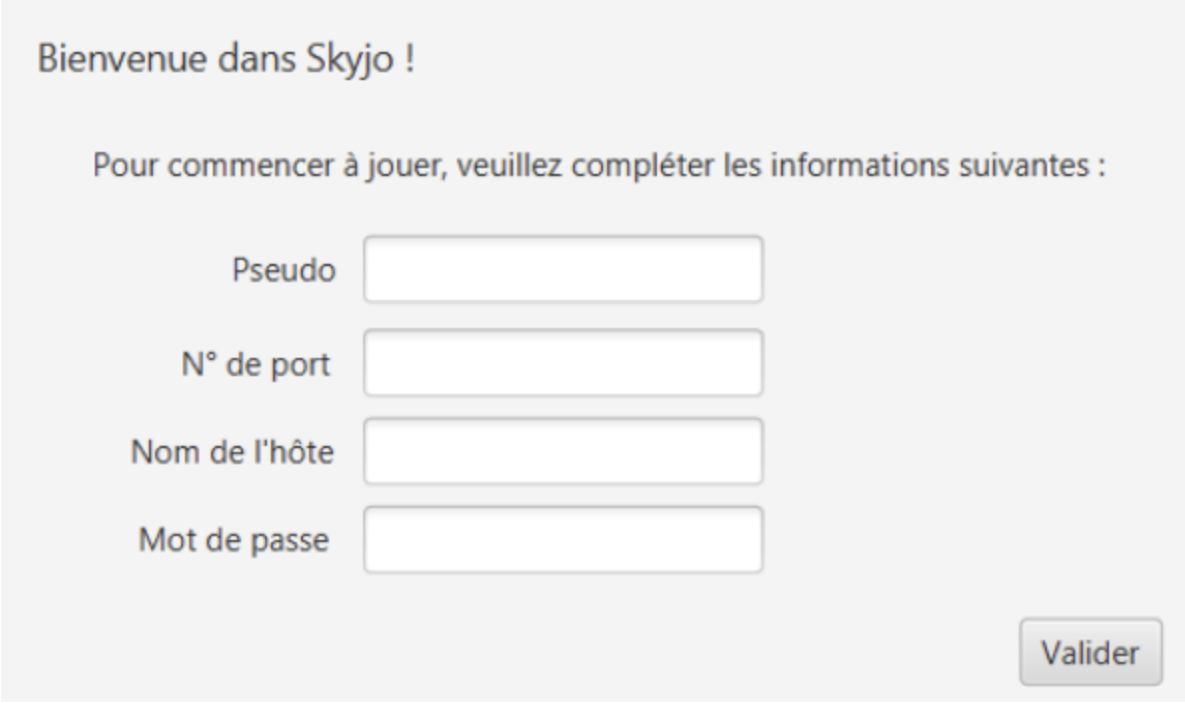
- FXMLConnectionController : classe controller FXML donnant la fenêtre reprenant les données de l'utilisateur.
- FXMLDisplayController : classe controller FXML d'une fenêtre.
- CardFX : représentant une carte en tant que bouton en javafx.
- GameFX : est la classe principale, qui donc possède une instance du model. Sa fonction principale est de coordiné en un temps les evénements des composant provenant des deux PlayerFx (GridPanes) ainsi que filtré intelligement les évenements venant de ceux-cis en fonction du statut du jeux.
- LayoutFX : est un observer qui étend le VBox et donc aura pour but d'ajouter l'HBox qui compose les deux GridPanes ainsi que la barre de status.

- MiddleFX : est un composant observateur étendant un GridPane qui contient deux CardFx, accompagnée de deux Buttons non actif dans le modèle que l'on place à coté des deux CardFx
  Les mises à jour se font indépendaments du model contrairement à un PlayerFx qu'on dois gérer en fonction du joueur courant.
- PlayerFX : est la classe qui représente les GridPanes composé d'un tableau fixe de CardFx qui representent les boutons.

  Cette instance permet de mettre à jour ces Buttons à chaque update c'est à dire de donnée des valeurs à ceux-cis.
- PlayerScoreFX : est une classe qui représente la barre de score de chaque PlayerFx, un playerFx étend celle-ci sur les 4 premières cellules du GridPane les mis à jour quand à elles sont fournies de GameFx.
- StatusBarFX : se situant dans LayoutFx est mis à jour en fonction des status que le model met à jour, celle-ci possède un String indiquant quel joueur est concerné par l'action (statut)
- TypePlayer : indique le joueur du client courant afin que la vue puisse être adaptée en fonction du joueur
- ViewFX : est un observer et à comme fonction principale de gérer la Stage et d'afficher une fenêtre d'informations en fonction des status de fin fu model
- L'interface AbstractClient : classe qui gère la communication avec le serveur
- Main : main du projet client
- SkyjoClientGui : contient toutes les méthodes de vue nécessaires
- TopMain : appel au main

Fenêtre FXML :

- FXMLConnexion.fxml : fenetre reprenant et demandant les données du client pour se connecter



- FXMLDisplay.fxml : fenêtre affichant une erreur

- FXMLWaitForOpponent.fxml : fenêtre d'attente attendant le 2ème client pour pouvoir démarrer une partie

SkyjoMessages :

Ce projet comporte les classes suivantes :

- MessageEndConnection : message à destination d'un client lui indiquant d'arrêter son processus
- MessageExistingConnection : message envoyé à un client si il tente de se connecter avec le même pseudo qu'un client déjà connecté lui indiquant d'arrêter son processus
- MessageExistingPseudo : message envoyé à un client qui tente de créer un pseudo déjà existant. Un message d'erreur est affiché
- MessagePasswordFalse : message envoyé à un client qui tente de se connecter avec un pseudo existant mais dont le mot de passe est erroné. Un message d'erreur est affiché.
- MessageSendEndGame : message contenant le winner,le score du gagnant,le score du perdant, à ajouter à la base de données ATLJ 6
- MessageProfile : message qui permet de mettre à jour les données du client, initialise, envoit le jeu aux joueurs et le jeu en cours aux spectateurs
- MessageSendModel : message qui envoit la dernière version du jeu au client non actif

- MessageWait : message permettant au client d'attendre la connexion de son adversaire
- Type (énumération) : les différents types des messages échangés
- Players: représentant une liste de tous les utilisateurs (joueurs) connectés
- User : représentant un utilisateur
  ainsi qu'une interface Message qui est utilisé pour tous les messages échangés.

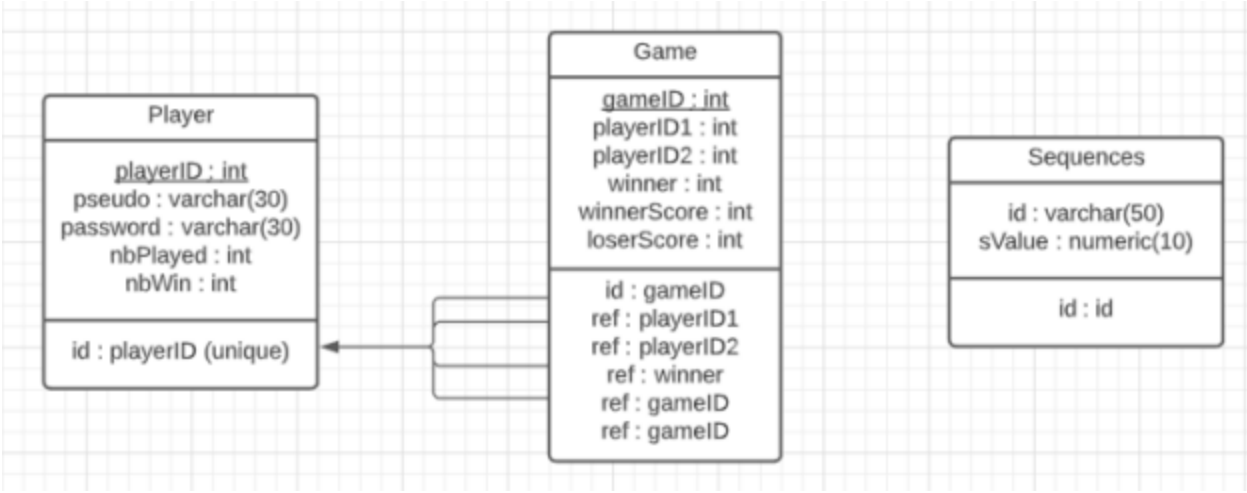
SkyjoServer :

Ce projet comporte les classes suivantes :

- SkyjoServerConsole : console pour le serveur
- L'interface AbstractServer : sert à écouter les demandes de connexion de la part des clients
- ConnectionToClient : classe créée par le serveur lorsqu'un client se connecte

Description d'une table SQL :

Voici le schéma conceptuel de notre base de données :



- *- Player*

Les joueurs de la partie. A partir des données récupérées via la fenêtre de login, un joueur sera inséré dans la table player. Un pseudo est unique. Si un client tente

de créer un joueur avec le même pseudo, une exception est lancée. Si un client tente de se connecter avec un pseudo existant et un mot de passe qui n'est pas lié à ce pseudo, une exception est lancée.

A la fin de chaque partie, le nombre de parties jouées et gagnées sont mises à jour.

- playerID : Identification du joueur
- pseudo : Le nom du joueur (unique)
- password : Le mot de passe du joueur
- nbPlayed : Le nombre de partie jouées
- nbWin : Le nombre de partie gagnées
- - **Game**

La partie de Skyjo. La game possède un id ainsi que l'id des 2 joueurs. A la fin de la partie, le gagnant, son score et celui du perdant sont stockés dans la base de données.

- gameID : Identification de la partie
- player1ID : Identification du joueur 1
- player2ID : Identification du joueur 2
- winner : Identification du gagnant
- winnerScore : Le score du gagnant
- loserScore : le score du perdant
- - **Sequences**

Pour chaque table de données, on a une table utilitaire (table sequences) qui permet de donner une ligne de départ.

- id : Identification de la séquence, la ligne de départ
- sValue : la valeur de départ générée