Wattier Alexandre Viho Juste Midedji Gardeur Nicolas
Rolecki Maciej El Fahsi Abdessalam

Rapport de projet

Description

Le principe du projet est de coder le jeu de carte Blackjack en utilisant Qt Creator. Nous avons décidé d'implémenter une interface graphique grâce aux outils fournis par Qt, tout en respectant le pattern MVC. Le projet a été organisé en client-serveur, cela signifie qu'il sera possible de jouer à plusieurs notamment grâce à l'introduction de threads. Le serveur embarque une connexion munie d'une database qui reprend les différentes informations des joueurs, comme leurs états en banque et les scores des différentes parties de jeu. Cela permet donc de sauvegarder les statistiques d'un joueur tout au long de la partie.

Bibliothèques utilisées

Nous avons utilisé la bibliothèque de Qt, une bibliothèque boost qui nous a permis de gérer les fichiers JSON, ceux-ci nous servaient à la communication entre le client et le serveur. Nous avons également écrit une librairie avec le modèle du jeu.

Mode d'emploi du projet

Le projet a été structuré en trois projets Qt :

- blackjack core
- blackjack server
- blackjack client

Afin de lancer le jeu, il faut d'abord que le serveur soit lancé. Néanmoins, les 2 projets devront être lancés de la même façon, c'est-à-dire sur Qt. Nous avons implémenté la connexion via une adresse de loopback pour le serveur. Au lancement du client, le joueur devra introduire son prénom (ce nom sera unique sur la DB mais 2 joueurs pourront utiliser le même afin de se connecter, il n'y a pas ce genre de vérification.) Par défaut un nouveau joueur débute avec 500 € sur son compte. Après insertion du nom, un simple appuie sur le bouton nous connecte donc à notre serveur.

Au lancement du jeu, la première option que le joueur aura est de miser sur son jeu. Afin de ne pas jouer, le joueur doit fermer sa fenêtre.

Après avoir choisi sa mise, le jeu vérifiera à chaque fois si tous les joueurs présents sont prêts pour joueur, c'est-à-dire qu'ils ont misé. Tant que ce n'est pas le cas, le jeu ne démarrera pas.

La communication graphique avec le joueur, c'est-à-dire les messages affichés dans les fenêtres se fais grâce à l'aide du serveur. Un update de la vue et des informations ne se fait que lorsque le client et le serveur échange des messages. Ensuite, le jeu se déroule classiquement comme une partie de BlackJack. Nous avons un croupier représentant la banque qui a pour but de distribuer des cartes aux joueurs. Le but étant de se rapprocher du chiffre 21 (grâce à l'addition des cartes) sans pour autant le dépasser.

La manche se finit si le joueur décide d'attendre ou dépasse 21. Si le joueur possède 7 cartes, là aussi c'est une victoire.

Le montant gagné est la mise multipliée par 1 ou 1,5 selon la façon dont il a gagné
Si l'on dépasse donc 21, nous perdons l'entièreté de notre mise.
Le joueur pourra donc jouer jusqu'à ce que le solde sa banque (Data Base) n'atteigne pas la somme équivalente à 0.

Répartition des taches

Chaque personne présente dans le projet a émis ses idées, et participé à l'élaboration de l'ensemble des packages. Certaines personnes ont notamment participé davantage sur certaines packages.
Client : Gardeur Nicolas / Rolecki Maciej
Serveur : Viho Midedji Just / Rolecki Maciej
Core : El Fahsi Abdessalam / Rolecki Maciej
Data Base : Wattier Alexandre / Rolecki Maciej
Rapport du projet : Gardeur Nicolas / El Fahsi Abdessalam

Description du projet

Le projet contient comme dit précédemment 3 projets :

- blackjackcore
- black_jack_server
- black_jack_client

Blackjackcore :
Il contient le modèle du game. C'est là ou vont se trouver les méthodes principales qui font que le jeu existe. On pourra aussi retrouver des classes permettant de structurer les objets sous format json pour la communication des paquets entre le client et le serveur.

Black_jack_server :
Pour commencer, c'est lui qui gère le game. Il prend en compte la communication entre le serveur et le client. C'est grâce aux fichier json, et à l'utilisation d'une énumération catégorisant les messages émis que celle-ci a été possible d'implémenter.
Il contient aussi la création de la base de données et ses requêtes. Nous avons utilisé le programme DB Browser for SQLite pour arriver à sa conception. Pour ce qui est des parties à plusieurs, il a fallu intégrer les threads que notre serveur prend donc en charge.

Black_jack_client :
Pour que le serveur puisse avoir une utilité, il faut bien avoir quelqu'un avec qui communiquer. Nous avons donc implémenté le client. C'est lui qui va discuter avec le serveur, donc lire/écrire des fichiers json avec lui. Après avoir reçu les informations du jeu, il faut bien obtenir quelque chose à l'écran. Le client comporte donc les 2 fichiers de la view : la connexion et la page du jeu. Celle-ci sera mise à jour au fur et à mesure du jeu.

Description des classes

Comme dit plusieurs fois auparavant, nous avons 3 package. Nous allons d'abord nous concentrer sur le premier package qui implémente la logique de notre jeu. Il s'agit du package core. Dans le package, nous retrouvons différents directory.

- Bj_json: Package qui contient tout ce qui concerne la gestion de json.

    - Json_intepreter : Cette classe permet d'interpréter les fichiers json
    - Json_gen : Cette classe permet de générer différents fichiers json.
    - Game_info: Cette classe permet la récupération des données dans le fichier json et de les assigner à des variables.
- Bj_packet_ip: Permet d'encapsuler le fichier json pour l'envoyer au serveur ou aux clients.
- Boost : Librairie qui permet l'implémentation de fichier json.
- Model :
    - Card : Une carte est composée d'une couleur et d'une valeur. Cette classe possède des méthodes basiques pour récupérer la valeur d'une carte.
    - Color : Enumération qui définit une couleur.
    - Croupier : Le croupier est la personne qui paie et encaisse l'argent du jeu. Le croupier hérite des méthodes d'un player.
    - Deck : Cette classe définie un deck. Un deck est composé de plusieurs cartes. Cette classe possède des méthodes pour créer un deck, mélanger le deck ainsi que y tirer une carte.
    - Game : Cette classe possède le déroulement du jeu avec différentes méthodes qui aident au déroulement de la partie. Cette classe possède une énumération qui donne le statut du jeu ainsi qu'une énumération de l'action effectuée qui est envoyée au serveur.
    - Player : Cette classe définit un joueur dans notre jeu. Un jouer possède les attributs suivant : le montant dans sa banque, sa mise, sa main … Les méthodes définies sont assez simples.
    - Value: Il s'agit d'une énumération qui a pour but de connaitre la valeur d'une carte.

Dans notre second package serveur, ce package permet la création d'un serveur. Il possède différentes classes :

- Db_connexion: Cette classe permet de créer la base de données SQL avec les différentes contraintes et liens entre ses tables, il permet aussi de récupérer et d'insérer les différentes informations d'un joueur et de sa partie.
- Game_handler : Cette classe permet de gérer son serveur et les diffèrents joueurs sur le serveur
- Player_thread :Cette classe permet de gérer les différents threads de joueurs
- Bj_tcp_server :C'est le serveur blackJack principal, il reçoit toutes les connexions avant de les rediriger vers le différents game_handler.

Dans notre dernier package client, ce package permet la création d'un client. Il possède différentes classes :

- Black_jack : Classe de la view principale du jeu. On peut y retrouver les méthodes permettant l'assignation des actions pour chaque bouton, mais aussi l'affichage des cartes pour les différents joueurs.
- Client_connection_window : C'est la classe de la view pour la connexion au jeu. On y retrouve par exemple le bouton qui permet de connecter le joueur au serveur, ou bien le texteEdit permettant d'inscrire le nom du joueur.
- Controller : C'est la classe principale du client. Elle rassemble chaque méthode nécessaire à la gestion du game et à la communication serveur client du côté du client. Par exemple les actions des buttons pour respecter le MVC. C'est là où se trouvent les méthodes read / write → essentielles pour les fichiers json, extraire les

données / en écrire pour répondre au serveur. Pour finir apply_game_info nous aidera à mettre à jour la view en fonction de l'action effectuée.

Erreurs

Les objets gérant la db nous retournent une Segmentation Fault dans le cas où un client rejoint, si on enlève la connexion à celle-ci, le client et le serveur tournent normalement.

Remarques

Lors de la compilation de la documentation, il faut faire attention à ne pas prendre la librairie boost car celle-ci possède beaucoup de fichiers documentés.

Pour que ça s'exécute, il faut d'abord compiler blackjack core et copier l'exécutable ainsi que les fichiers sources dans bj_lib/blackjackcore du client et du serveur.

Table des matières