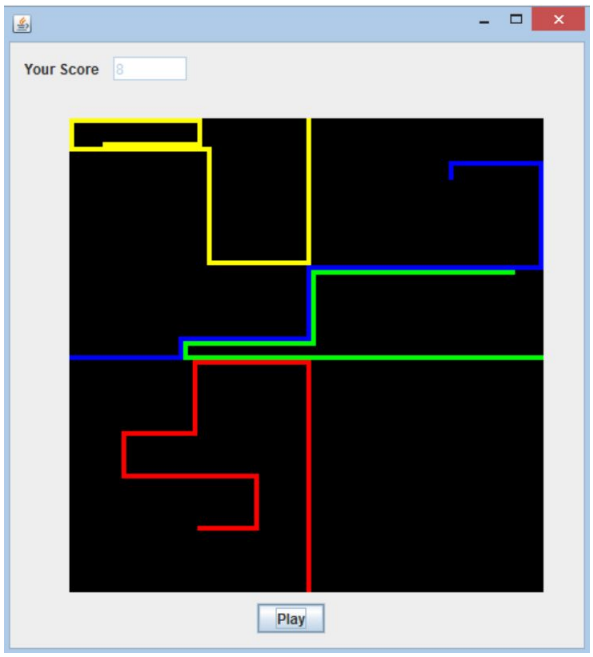
DÉPARTEMENT TECHNIQUE DE MARCHE-EN-FAMENNE
MASTER EN **A**RCHITECTURE DES **S**YSTÈMES **I**NFORMATIQUES



[Samuel HIARD - MASI Henallux]

Programmation d'applications distribuées et en réseau

Rapport : Projet d'évaluation 2

Réalisé par Johan OREEL et Luther DIANGA,
Étudiants du MASI

INTRODUCTION

Les outils utilisés pour la réalisation du projet

- Système d'exploitation : Mac OS et Ubuntu 16.4
- Netbeans IDE 8.1 et IntelliJ IDEA 2016.3
- Java 8

MÉTHODES ET CLASSES

////////////////////////////////// CLASSES

Account :

Classe contenant le login et le mot de passe d'un joueur. Elle permet aussi de savoir si un joueur est en ligne ou non. Cette classe ne contient pas de méthode, mais des getters & fonctions :

- boolean isOnline() : Retourne si le joueur est en ligne
- void setOnline(boolean online) : met le jouer en ligne ou hors ligne

AIPlayer :

Classe représentant un joueur contrôlé par une intelligence artificielle.
fonctions :

- IAI getAi() : retourne l'intelligence artificiel qui contrôle le joueur.
- void setAI(IAI ai) : définit l'intelligence artificiel qui contrôlera le joueur.

AlreadyLoggedInException :

Une exception qui doit être lancée lorsqu'un utilisateur qui est déjà connecté tente de se connecter via un autre client.

ClientSession :

Classe permettant de regrouper l'ID du joueur, le compte et le stub d'un joueur. Elle permet aussi de garder une trace du lobby dans lequel le joueur joue.
fonctions :

- **void** incrementConnectionFailures() : Augmente le nombre d'échecs de connexion à 1.
- **void** resetConnectionFailures() : Réinitialise le compte de défaillance de connexion à 0.
- **boolean** isDeclaredDead() : retourne si le nombre de connexions échouées a dépassé le nombre permis.
- **void** leaveGameLobby() : quitte le lobby actuel.

Core :

C'est la classe qui gère tous les calculs d'une session de jeu. Cette classe est responsable de garder l'état du jeu ainsi.
fonctions :

- **void** runGame() : commence le jeu
- **void** endGame() : met le drapeau à fin du jeu
- **boolean** isGameInProgress() : retourne si le jeu est toujours en cours
- **void** newGrid() : recrée la situation de départ

Player :

Classe permettant d'assembler tous les attributs d'un joueur dans une classe abstraite.
fonctions :

- **void** incrementIxCarPos() : incrémente la position x
- **void** decrementIxCarPos() : décrémente la position x
- **void** incrementIyCarPos() : incrémente la position y
- **void** decrementIyCarPos() : décrémente la position y

HumanPlayer :

Un joueur contrôlé par une personne utilisant les touches fléchées.
fonction :

- **void** incrementScore() : incrémente le score de 1.

GameLobby :

Cette classe gère le jeu en mode multijoueurs.
fonctions :

- **void** startGame() : lance le jeux
- **void** startTimer() : lancer le timer en multijoueur
- **boolean** join(**long** clientID) : Ajoute un joueur à la liste s'il peut encore participer au jeu
- **void** leave(**long** clientID) : Supprime un joueur du multijoueur
- **void** setCurrentDirection(**long** clientID, **char** newDirection) : Modifie la direction actuelle du joueur.
- **char** getCurrentDirection(**long** clientID) : retourne la direction actuelle du joueur.
- **boolean** isPlayerAlive(**long** clientID) : retourne si le client est encore en vie ou pas.

GUI :

Il s'agit de l'interface graphique utilisateur (GUI). Son rôle principal est d'afficher l'état du jeu, mais aussi de transmettre des informations au processus central.
fonctions :

- **void** showServerNotFoundDialog() : Montre un message d'erreur si le serveur est introuvable.

GuiLogin :

C’est l’interface graphique utilisateur pour s’inscrire ou se connecter.
fonctions :

- **void** signIn() : Tente de s’inscrire en utilisant les informations d'identification.
- **void** logIn() : Tente de se connecter à un utilisateur en utilisant les informations d'identification.
- **boolean** isFormFilled(String login, String pwd) : Vérifie si l'une des chaînes donnée est vide.

GuiMultiplayer :

C’est la fenêtre principale pour le mode multijoueur. Dedans nous retrouvons une liste des utilisateurs connectés et à la fin d’une partie le jeu se lance avec les utilisateurs.
fonctions :

- **void** close() : ferme la fenêtre multijoueur.
- **void** leaveGameLobby() : quitter le lobby du jeu actuel.
- **void** updatePlayerList(String[] playerNames) : met à jour la liste d’utilisateur.
- **void** updateTimer(**int** countDown) : met à jour le compteur.

RmiClient :

C’est la classe qui implémente l’interface client.
fonctions :

- **void** hello() : permet au serveur de voir si le client est toujours en ligne.
- pdateGame(**int** score, **int**[][] grid, **boolean** isGameOver, String winnerName) : met à jour l’écran de jeu.
- **void** updateLobbyPlayerList(Collection<String> playerList) : met à jour la liste des utilisateurs pour l’écran en multijoueur.
- **void** updateLobbyTimer(**int** countDown) : met à jour le Timer pour l’écran en multijoueur.
- **void** joinGame(**boolean** solo) : Lance le jeux en solo ou rejoint un salon pour le multijoueur.
- **void** quitGame() : quitte le salon multijoueur.
- **void** quitApp() : quitte le jeu et se déconnecte du serveur.
- **boolean** isPlayerAlive() : retourne si le joueur est toujours vivant dans la session actuelle.
- **boolean** createAccount(String login, String pwd) : interroge le serveur pour créer un nouveau compte.
- logIn(String login, String pwd) : Tentative de connexion au serveur.

RmiClientMain :

C’est la classe main du client, elle permet de lancer le client.

RmiServer :

Cette classe implémente l'interface distante.
fonctions :

- **void** helloService() : vérifie à chaque intervalle si le login de l'utilisateur est toujours connecté.
- **synchronized void** removeClient(**long** clientID) : supprime le client de la liste.
- **void** updateLobbyPlayerList(Collection<Long> clientIDs) : Demande aux clients concernés de mettre à jour leur liste de joueur en multi.
- **void** updateLobbyTimer(Collection<Long> clientIDs, **int** countDown) : Demande aux clients concernés de mettre à jour leur compteur en multi.
- **void** updatePlayer(**long** clientID, **int** score, **int**[][] grid, **boolean** isGameOver, String winnerName) : demande au client de mettre à jour leurs écrans de jeu.
- **synchronized boolean** createAccount(String login, String password) : essaie de créer un compte.
- **synchronized** Long logIn(IClient client, String login, String password) : essaie de connecter un joueur.
- **synchronized void** logOut(**long** clientID) : déconnecte un joueur du serveur.
- **synchronized void** joinGame(**long** clientID) : Permet à un joueur de rejoindre une partie en multi.
- quitGame(**long** clientID) : permet au client de quitter le salon multijoueur.

RmiServerMain :

C'est la classe qui permet de lancer le serveur.
fonction :

- **void** startRegistry(**int** RMIPortNum) : Cette méthode lance le RMI registry en localhost, s'il n'existe pas déjà sur le numéro de port spécifique.

CHOIX D'EN L'APPLICATION

Nous avons décidé de séparer le projets côté client en trois parties. Le premier écran est le login. Il permet à l'utilisateur de soit s'authentifier sur le serveur soit créer un nouveau compte. Une fois authentifié, le deuxième écran s'ouvre, l'écran du jeu. Sur cet écran, on peut voir les options solo et multijoueur.

Si l'on choisit le mode solo, une partie commence en local contre trois AI. Si on choisit multijoueur, la troisième fenêtre s'ouvre, le lobby. Dans le lobby les joueurs attendent leurs adversaires. Quand au moins 2 personnes se trouvent dans le même lobby et que le timer arrive à 0, la partie commence. Les places vides sont remplie par des AI. Cet partie est géré par le serveur.

Au bas gauche de l'écran de jeu est marqué le nom du joueur et la couleur qu'il joue dans la partie. Un joueur est capable de quitter une partie en cours à tout moment en cliquant à nouveau sur play. Les autres joueurs avec qui il jouait, continue leur partie sans se rendre de compte du départ d'un joueur.

Le serveurs vérifie de façon régulière si les joueurs connectés sont toujours là, ainsi si un joueur devrait disparaître de façon imprévu, le serveur est au courant et marque que le joueur s'est déconnecté.

MODE D'UTILISATION

Dans le dossier se trouvent plusieurs sous-dossiers, "client", "javadoc" et "server". Dans "client" se trouvent toutes les sources java du client avec le makefile, le ".sh" et les ".bat". Dans "server" se trouvent les fichiers nécessaires au serveur. Dans "javadoc" se trouvent la documentation java.

Les "makefile" et les ".sh" ont été testé sur un Ubuntu 16.04 et les ".bat" sur un Windows 10. Sur Windows 10 il faut d'abord vérifier que la variable d'environnement "Path" soit bien configurée avec le lien vers le jdk.

Dans le dossier "server" le fichier "server.sh" sur Linux et "server.bat" sur Windows permettent de lancer la partie serveur. Il faut donner en paramètre l'adresse ip de la machine serveur et éventuellement le port RMI. Dans le dossier "client" le fichier "client.sh" sur Linux ou "client.bat" sur Windows permet de lancer la partie client. Le fichier "index.html" dans le dossier "javadoc" ouvre l'index de la documentation.

PROBLÈMES RENCONTRÉS

Nous avons rencontré énormément de problèmes concernant la connectivité entre plusieurs machines physiques et virtuelles. Pourtant le projet marche parfaitement sur en localhost. Une fois que le client se retrouve sur une autre machine que le serveur, le client arrive à contacter le serveur, mais le serveur n’arrive pas à le répondre.