Rapport de projet

# Description

Le principe du projet est de coder le jeu de carte Skyjo en utilisant java avec une interfaces graphique à l’aide de java fxml, tout en respectant le pattern MVC et en implémentent le principe d’observateur observé.

Skyjo est un jeu dont l’objectif est d’obtenir le moins de points possibles en inversant et échangeant des cartes. Chaque joueur possède 12 cartes dont la valeur varie de -2 à 12. Jouable de 2 à 7 joueurs. La description des règles du jeu est disponible via <https://www.regledujeu.fr/regle-du-skyjo>.

## Bibliothèques utilisées

Nous avons utilisé la bibliothèque openjfx et 2 librairies personnalisées Message\_Protocol\_SKYJO et model\_skyjo que l’on a écrites.

# Règle du jeu

## Début d’une partie

Lorsqu’une manche démarre, deux cartes ont été retournées pour chaque joueur. Le joueur avec le plus de points commence. Si les deux joueurs ont le même nombre de points, choisissez au hasard.

## Déroulement d’une manche

Le premier joueur doit choisir entre les deux actions suivantes :

1. Prendre une carte dans la défausse.

* Si le joueur choisit de piocher une carte de la défausse, il doit échanger la carte défaussée avec une de ses cartes. Il peut l’échanger contre n’importe laquelle de ses cartes, qu’elle soit face visible ou face cachée. Lorsqu’il procède à cet échange, il rend la carte échangée face visible sur la pile de la défausse.

1. Prendre une carte de la pioche

* Si le joueur choisit de retourner une carte de la pioche, il peut
  + Soit reposer la carte tirée sur la pile de défausse (face visible) et retourner une de ses cartes dont la valeur était cachée.
  + Soit il peut échanger la carte piochée avec n’importe laquelle de ses cartes. Lorsqu’il procède à cet échange, le joueur dépose son ancienne carte face visible sur la défausse.

Après cette action, le nombre de points du joueur est mis à jour et le joueur suivant peut jouer.

## Fin d’une manche

Une manche se décompose en plusieurs tours. Chaque tour permettant aux joueurs d’effectuer leurs actions. Une manche se termine lorsqu’un des joueurs a retourné toutes ses cartes. Quand le premier joueur du tour a retourné toutes ses cartes, les autres joueurs auront encore droit à effectuer une action chacun. C’est seulement à la fin de ce tour que la manche est terminée.

## Fin du jeu

Si le joueur qui a terminé en premier a un score plus grand ou égal au score de l’autre joueur et si son score est positif, une pénalité lui est attribuée : ses points sont doublés.

# 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 parties :

* Wattier Alexandre : Model / View / Rapport Projet
* El Fahsi Abdessalam : View / Model / Doc
* Rolecki Maciej : Client / Serveur / Model / View / Base de données
* Viho Juste Midedji : Client / Serveur / View / Model / Librairie
* Gardeur Nicolas : Model / Client / Doc / Base de données

# Description des classes

Le projet a été séparé en 3 parties pour une meilleur visualisation : la partie MODEL, la partie SERVEUR et la partie CLIENT.

## Pour la partie MODEL

Ce package reprend l’ensemble des classes, et donc des méthodes nécessaires au bon déroulement du jeu. Le jeu sera stocké sur le serveur pour une meilleur fonctionnalité du jeu.

## Pour la partie SERVEUR

Ce package reprend la classe liée à la fonctionnalité du serveur.   
C’est ce package qui va servir d’écrous centrale au projet, car c’est là qu’on se connectera depuis le client, mais là aussi où l’on récupérera les informations souhaitées.  
 Pour que cela se déroule du meilleur des manières, une libraire a été ajouté par Juste.

## Pour la partie CLIENT

Ce package reprend les classes liées à la fonctionnalité du client. C’est là que les utilisateurs vont pouvoir se connecter et ainsi lancer le jeu.

# Description des classes niveau serveur

Le package motherServer qui contient la classe primaryServer qui étend thread et qui utilise plusieurs attributs dont le gameServersId qui permet de connaitre le nombre de jeu qui tourne, une liste des serveurs secondaires qui englobe l’ensemble des instances de l’objet secondaryServer gérant les jeux en cours, le port utilisé et un Boolean. Il initialise le serveur secondaire avec un nouveau jeu et redirige les clients vers le serveur secondaire (redirection socket client vers thread adéquat).

Le package secondaryServer contient la classe playerThread qui étend thread et représente le thread d’un joueur connecté. Il utilise plusieurs attributs qui sont :

* ClientSocket
* User (Les informations sur le joueur)
* SecondaryServer (l’instance gérant l’objet Game)
* Un InputStream et Outputstream :

Il peut établir la connexion avec les joueurs, il peut démarrer la communication avec les différents joueurs, couper la connexion avec eux, mettre à jour le score des joueurs et envoyer un message à un client en utilisant un protocole implémenté.

Le package contient aussi la classe secondaryServer qui contient plusieurs variable dons le nombre minimum de joueur et le nombre maximum de joueur mais aussi le thread du premier joueur et second joueur.

# Façade du Game

* SrvId (id du serveur secondaire)
* Full : Boolean, pour savoir si le serveur secondaire est plein
* Une copie du serveur principal
* Un accès à la base de données
* Id des membres

La classe va gérer les données reçues des clients :

* Vérifier si la partie de jeu est lancée
* Créer un user et l’ajouter à la DB
* Ajouter un utilisateur sans jeu créé
* Envoyer un message de type message créé par l'utilisateur aux nouveaux utilisateurs
* Créer et envoyer à l'utilisateur administrateur que l'utilisateur est prêt et que le jeu peut être démarré
* Créer un nouvel objet de Game et l’envoyer à tous les joueurs en ligne.

Le package launcher contient la classe launcherSrv qui contient le main de notre serveur.

## Description de la partie model

Elle contient le package be.he2b.atlr5.skyjo.model qui contient les classes

* Card : Crée une instance de carte avec une valeur une visibilité et un id.
* Deck : Crée une instance du deck et de la défausse et les enregistre, on peut retirer une carte en début du deck ou en fin de la défausse, retourner 12 cartes (pour le début du jeu).
* Façade : c’est la façade du Game avec la méthode qui peuvent être utiliser dans la vue.
* Game : crée une instance de lui-même et va faire appel à la création du deck, des joueurs concernés, de la liste de joueur qui les contiendra et crée une carte temporaire afin de contenir celle que les joueurs choisissent. On retrouve les méthodes pour mettre à jour le score des joueurs et mettre fin à la partie. La classe permet également de jeter, retourner une carte, tirer une carte et décider qui commence et mettre à jour les joueurs.
* Player : Crée une instance de Player avec une liste de cartes pour la main du joueur, son score, son état (playing, fail ou win), la carte choisie et les détails du joueur (id, mail, id dans la DB).
* PlayerChoice : énumération qui contient les diffèrent choix que le joueur peut faire (piocher et jeter, piocher et garder, prendre dans la défausse).
* PlayerDetails : crée une instance d’elle et contient l’Id dans le jeu courant, le mail et l’Id du joueur enregistré dans la DB.
* State : énumération qui contient les diffèrents état que le joueur peut avoir (playing, fail ou win).
* Value : énumération qui contient les différentes valeurs des cartes (de –2 à 12).

## Description de la partie client

Elle contient le package be.he2b.atlr5.Skyjo qui contient 4 packages

* ClientConnexion : étend abstractClient et a comme attributs
  + user
  + La liste des messages que le client peut envoyer au serveur. Il gère la connexion niveau client c'est lui qui reçoit les messages du serveur et le redirige vers la view pour la mise à jour
* Controller va servir de lien entre la façade game, la vue et la connexion client au serveur
* Main Package contient la classe MainFunction
  + MainFunction étend application et sert à démarrer l’interface du client
* View contient 2 packages, le package viewJfx et buttonHandlers
  + viewJfx contient les classes CardButton, CardLabel, GeneralMethod, Leftpane, LocalLabel, MiddleVBox ,PlayerVBox , RightPane, SkyjoHBox , SkyjoVBox
  + ButtonHandlers contient les classes ChoiceBoxHandler, s'il change de valeur une notification sera envoyé au serveur en prenant des informations
    - DropButtonH implémente eventhandler et permet de gérer si le joueur peut cliquer ou non sur le bouton.
    - HandlerCardB implémente eventhandler et permet de gérer si le joueur peut cliquer ou non sur ces cartes
    - HandlerDiscardB implémente eventhandler et permet de gérer si le joueur peut cliquer ou non sur le bouton défausse
    - HandlerPickB implémente eventhandler et permet de gérer si le joueur peut cliquer ou non sur le bouton pioche
    - KeepbuttonH implémente eventhandler et permet de gérer si le joueur peut cliquer ou non sur le bouton garder

# JDBC

Pour l’implémentation de la base de données, on a utilisé SQLite. Afin de créer nos différentes tables, nous avons utilisé le logiciel DB Browser for SQLite : <https://sqlitebrowser.org/>

Dans notre projet, on a intégré 3 tables différentes qui sont :

* Nom : USER
  + Attributs : ID, MAIL
* Nom: DATA\_USER
  + Attributs: ID\_USER, NBR\_WINS, NBR\_GAMES, LOWEST\_SCORE
* Nom: DATA\_GAMES
  + Attributs: ID\_GAME, ID\_USER, SCORE

Nous utilisons user afin d’effectuer une connexion, en effet, nous vérifierons si la personne souhaitant se connecter fais partie de la base de données ou non. Il faudra donc avoir la possibilité d’en créer si c’est un nouveau joueur.

La table DATA\_USER va récolter les infos propres d’un user concernant les parties. Par exemple je vais récupérer son nombre de parties gagnées, ou voir son nombre de parties aux quels il a participé. Il faudra donc à chaque fin de partie mettre les données à jour.

Et la table DATA\_GAME va elle récupérer le score des joueurs d’une partie. Grâce à elle, on va pouvoir récupérer le score des joueurs dans une partie précise dû à l’ajout de ID\_GAME.

Nous utiliserons donc une suite de Select / Insert / Update dans nos tables.

Nous avons utilisé et modifié les classes fournies par Mme Rousseau car celles-ci étaient déjà fonctionnelles.

# Parties non fonctionnelles

Actuellement, la mise à jour de la table Data\_Game de la DB ne fonctionne pas, la commande pour le faire a bien été écrite mais un bug est lancé en fin de partie de jeu.

# Modification second session

Ajout de la classe DAO   
Modification de maven à ant pour utiliser   
Soucis obtenu lors de l’update (réglé): DB bloqué

Contents

[Description 1](#_Toc90241044)

[Bibliothèques utilisées 1](#_Toc90241045)

[Règle du jeu 1](#_Toc90241046)

[Début d’une partie 1](#_Toc90241047)

[Déroulement d’une manche 1](#_Toc90241048)

[Fin d’une manche 2](#_Toc90241049)

[Fin du jeu 2](#_Toc90241050)

[Répartition des taches 2](#_Toc90241051)

[Description des classes 2](#_Toc90241052)

[Pour la partie MODEL 3](#_Toc90241053)

[Pour la partie SERVEUR 3](#_Toc90241054)

[Pour la partie CLIENT 3](#_Toc90241055)

[Description des classes niveau serveur 3](#_Toc90241056)

[Façade du Game 4](#_Toc90241057)

[Description de la partie model 4](#_Toc90241058)

[Description de la partie client 5](#_Toc90241059)

[JDBC 6](#_Toc90241060)

[Parties non fonctionnelles 7](#_Toc90241061)