2018

Monopoly HEIG-VD

CAHIER DES CHARGES

BURGENER, CURCHOD, GONZALEZ LOPEZ, REYMOND

7 mai 2018 Version : 1.250

Monopoly HEIG-VD

Table des matières

Descriptif

Dans le cadre du module de Génie Logiciel (GEN), il nous est demandé de développer un "mini-projet" implémentant une architecture client-serveur pouvant communiquer par internet en utilisant les sockets. Après discussion nous sommes tombés d'accord sur l'idée d'un projet Monopoly sur le thème de la HEIG-VD.

Fonctionnement Général

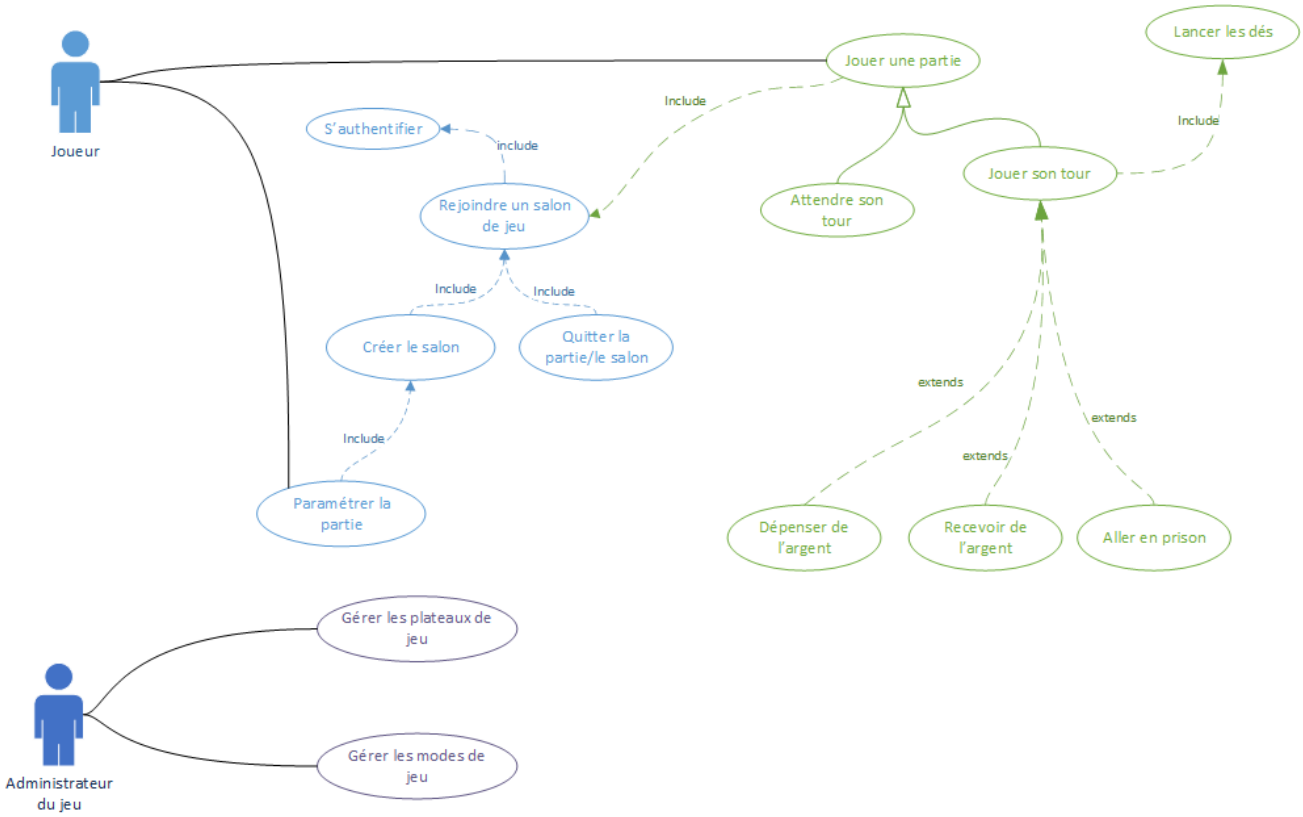
Pour notre jeu de société, voici les fonctionnalités de base que nous avons choisies de mettre en place :

- **Authentification simple** : nous demandons un nom d'utilisateur ainsi qu'un mot de passe à la connexion de l'utilisateur. Si le nom d'utilisateur n'est pas reconnu, nous proposons de créer un compte utilisateur avec ces informations.
- **Création d'un salon de jeu** : N'importe quel utilisateur peut créer son salon de jeu. Cela lui permet de devenir le gérant d'une partie, et donc de choisir les différents paramètres de la partie qui réglementeront le déroulement du jeu.
- **Possibilité de rejoindre un salon de jeu** : l'utilisateur aura à sa disposition une liste des salons de jeu qui sont en attente de joueur pour commencer la partie.
- **La phase de jeu** : le joueur pourra lancer les dés (lors de son tour), choisir d'acheter un terrain ou non, payer la caution pour la sortie de la salle d'examen ou tenter le lancer de dés. Sinon il devra se plier aux "effets" des cases sur lesquelles il arrivera (CF règles du monopoly).
- **Espace administrateur** : l'administrateur peut, dans une interface différente, modifier certains paramètres concernant les limites du jeu. Par exemple il pourrait fixer les limites du nombre de dés utilisés et du capital de départ autorisé. Mais il aurait également la possibilité d'activer/désactiver certains paramètres de jeu, qui deviendraient ainsi visibles dans l'interface de création de partie des joueurs (exemple : permettre ou non la génération aléatoire du plateau de jeu).
- **Suivi des scores et statistiques** : une fois créé, l'utilisateur peut en tout temps voir ses différents scores réalisés sur les parties jouées (date de début, date de fin, état d'abandon de la partie, argent en poche à la fin de la partie, nombre de bâtiments achetés, nombre de visites en salle d'examen, nombre de cases parcourues).

Les fonctionnalités ci-dessus se concentrent principalement sur le point de vue du joueur. Nous considérons que cette partie de notre projet est primordiale et doit passer impérativement avant une éventuelle implémentation des fonctionnalités optionnelles listées ci-dessous. Nous implémenterons ces dernières si le temps nous le permet :

- **Messagerie instantanée** dans le salon de préparation et lors de la partie
- **Ajout d'options dans la partie administrative du jeu** :
    - Vue globale des salons en cours
    - Génération aléatoire du plateau de jeu
    - Gestion du plateau de jeu
    - Nombre de cases « Aller en salle d'examen »
    - Modifier le prix des propriétés (ratio de modification, ex : 1.1x plus cher)
    - Limite de temps pour une partie
- **Interaction avec les autres joueurs** (échanges/troc)

Diagramme des cas d'utilisation



Acteurs

Nous avons prévu 2 acteurs pour nos scénarios :

- L'administrateur du jeu qui peut gérer la disposition des plateaux de jeu, ainsi que les "modes" de jeu (autre que classique, des paramètres personnalisés)
- Le joueur qui sera le principal acteur de notre projet. C'est lui qui participera à une partie de Monopoly, mais pour cela il devra tout d'abord s'authentifier, puis choisir une partie. Enfin il attendra que tous les joueurs soient prêts, pour que la partie se lance. En jeu, il devra attendre son tour pour lancer les dés et être actif dans la partie.

Scénario principal

Voici comment nous voyons le déroulement d'une partie de Monopoly :

Scénario de préparation

Le scénario de préparation englobe les actions que le joueur devra entreprendre avant de pouvoir jouer.

L'utilisateur va **s'authentifier** à l'aide de son nom d'utilisateur et son mot de passe, son compte sera créé en cas de nom inconnu dans la base de données.

L'utilisateur authentifié pourra parcourir la liste des parties en attente de joueurs dans le but de **rejoindre un salon de jeu**.

Il pourra également **créer son propre salon de jeu** pour y accueillir d'autres joueurs, cela lui permettra de **modifier les paramètres de la partie**.

Une fois que le joueur se sent d'attaque à commencer la partie, il peut se **déclarer "prêt"** et attendre que les autres joueurs fassent de même. Une fois tous les joueurs prêts à jouer, la partie se lance.

Scénario de jeu

Une fois la partie lancée, les participant passent en phase de jeu.

Avant de pouvoir jouer, le participant devra **attendre son tour**. Pendant ce temps il peut inspecter les cases du plateau.

Une fois son tour arrivé, il devra **lancer les dés**, pour avancer dans le plateau et ainsi participer à la partie. En cas d'inactivité, son tour sera automatiquement passé. Si le joueur fait un double (les deux dés ont la même valeur), le joueur devra refaire un tour (relancer les dés). Au bout du troisième double de suite, il sera contraint d'**aller en salle d'examen**.

Durant la partie, le joueur aura de multiples occasions pour **dépenser son argent**, comme l'achat de propriétés ou encore payer un loyer/une facture, ou pour **en recevoir**, comme la réception d'un loyer ou la vente d'une propriété.

Interfaces graphiques

Voici un aperçu de notre projet, tel que nous l'imaginons :

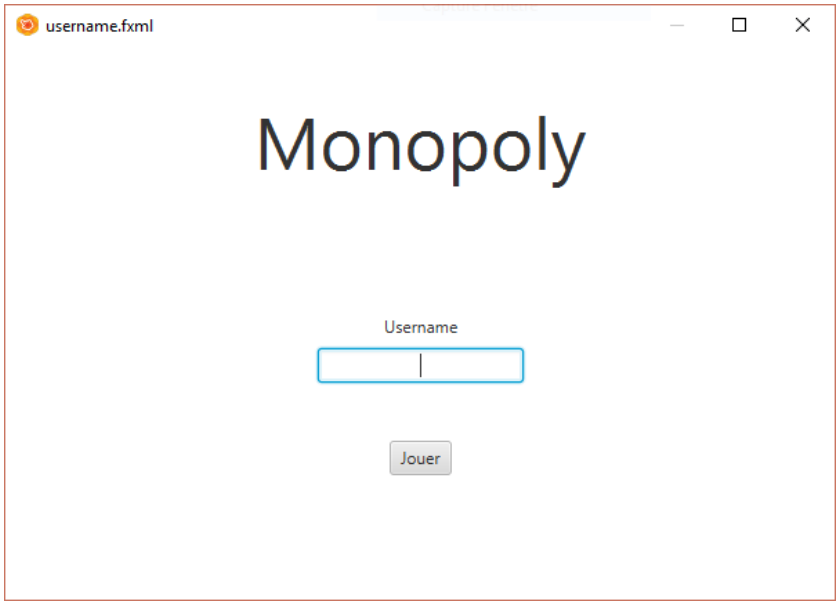


Figure 1 - fenêtre de connexion

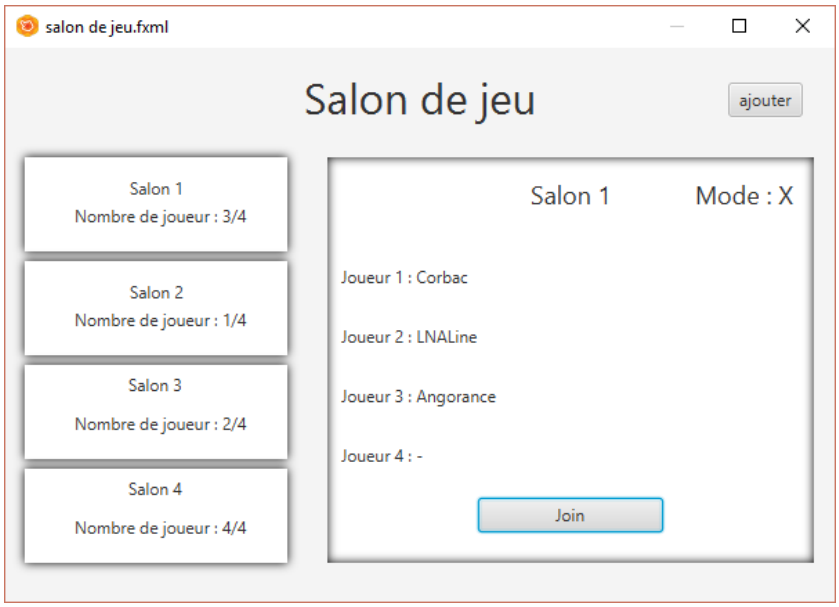


Figure 2 - Liste des salons de jeu

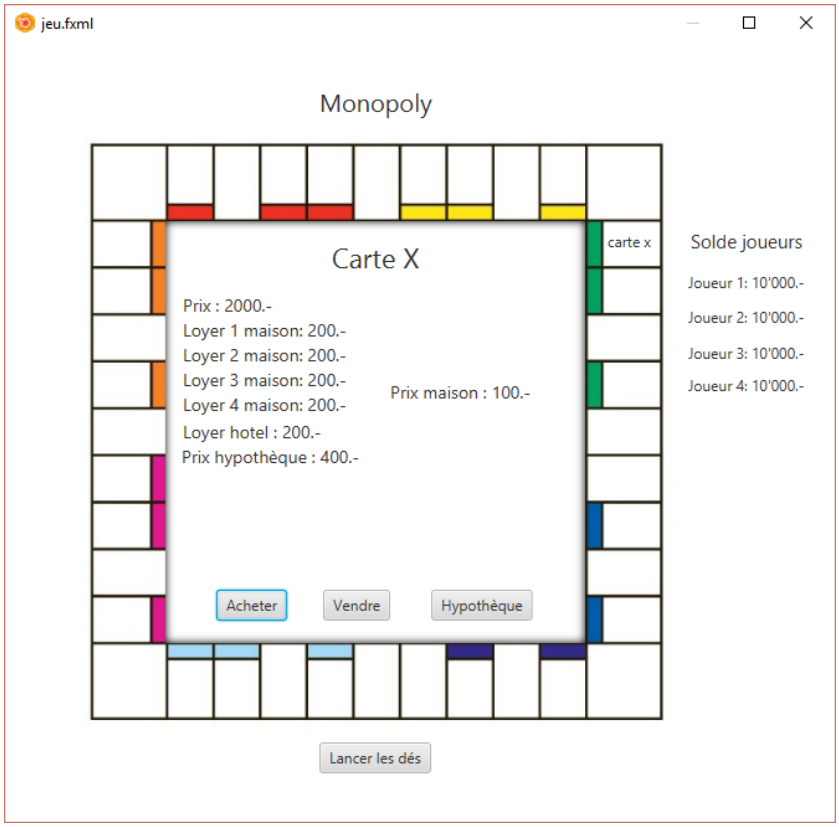


Figure 3 - Phase de jeu

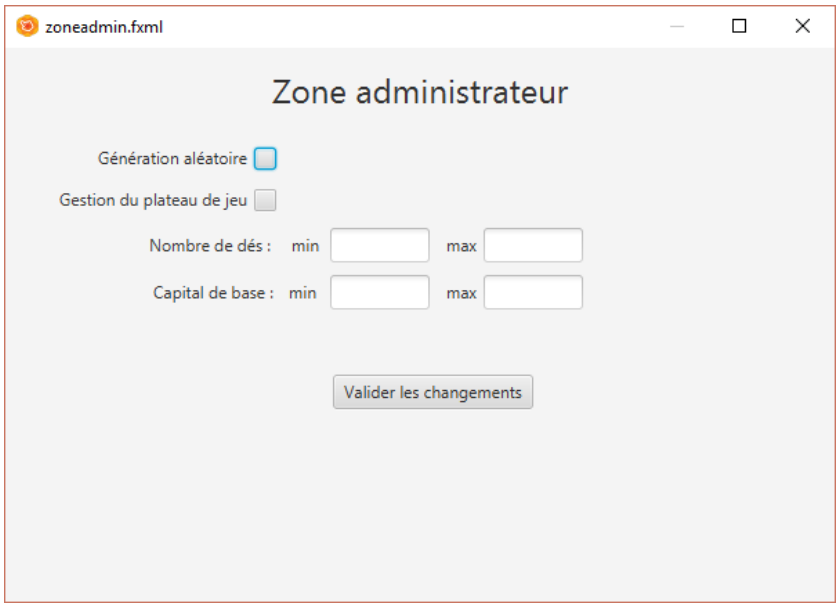


Figure 4 – Zone de l'administrateur

Règles du jeu

Pour notre projet, nous nous inspirons des règles officielles du jeu Monopoly.

Partage des responsabilités entre le serveur et le client

Afin d'éviter une surcharge de communications vers le serveur, nous avons décidé que toute la logique du jeu serait implémentée du côté serveur. C'est-à-dire que le serveur implémentera tout ce qui touche aux transactions, aux échanges et accords entre les joueurs, au calcul des statistiques, à la génération du plateau de jeu et des decks de cartes communautaire/chance.

Le client, de son côté, s'occupera d'afficher le plateau de jeu et de maintenir à jour l'état de son joueur (exemple : il dira au serveur de déplacer le joueur A de X cases, si le joueur est en prison, etc.).

Protocole d'échange entre le client et le serveur

Les échanges entre le client et le serveur sont principalement séparés en deux parties : une phase login-lobby qui consistera en la gestion de l'authentification de l'utilisateur, ainsi que la gestion des salons (et leur création) et une phase de jeu. Pour le moment nous avons simplement fourni le protocole utilisé pour la première phase, bien que ce protocole soit sujet à d'éventuelles modifications. Nous ne fournissons pas de protocole pour la seconde partie, car nous n'avons pas encore la vision nécessaire pour fournir une structure convenable.

Pour ce qui est des outils que nous utilisons pour la communication entre le client et le serveur, nous avons choisis d'opter pour des sockets en ce qui concerne les communications sur le réseau. Enfin nous devrons partager des informations sur des objets, nous utiliserons donc la sérialisation à l'aide de JSON.

Voici donc le protocole applicatif que nous utiliserons pour cette première phase :

Message	Effet
Depuis le client	
LOGIN [username] [password_digest]	Demande au serveur si un compte avec ces identifiants existe
RGSTR [username] [password_digest]	Demande au serveur s'il est possible de créer un compte avec ces identifiants
JOIN [lobby_id]	Tente de rejoindre un lobby s'il y a assez de place
NLOBBY	Crée un lobby, cette commande est suivie des informations sur le salon de jeu à créer (sérialisation d'un objet s'apparentant à un salon de jeu), puis de la commande JOIN
READY	Se déclarer prêt à commencer la partie
Depuis le serveur	
OK	Opération possible, confirmation (réponse aux commandes LOGIN, RGSTR et JOIN)
UNKNOWN	Aucun compte utilisateur correspondant trouvé (réponse à LOGIN)
DENIED	Mot de passe erroné (réponse à LOGIN) ou identifiant déjà utilisé (réponse à RGSTR)
START	Tous les joueurs sont prêts, la partie va commencer

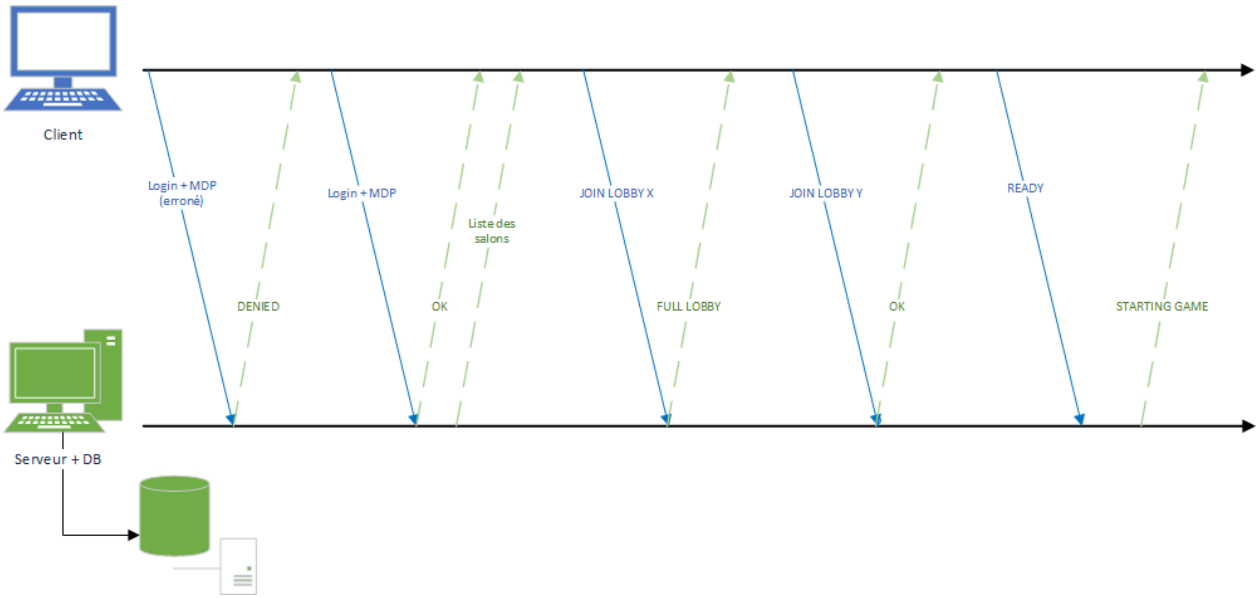


Figure 5 - schémas de communication : phase login-lobby

Modèles de domaines

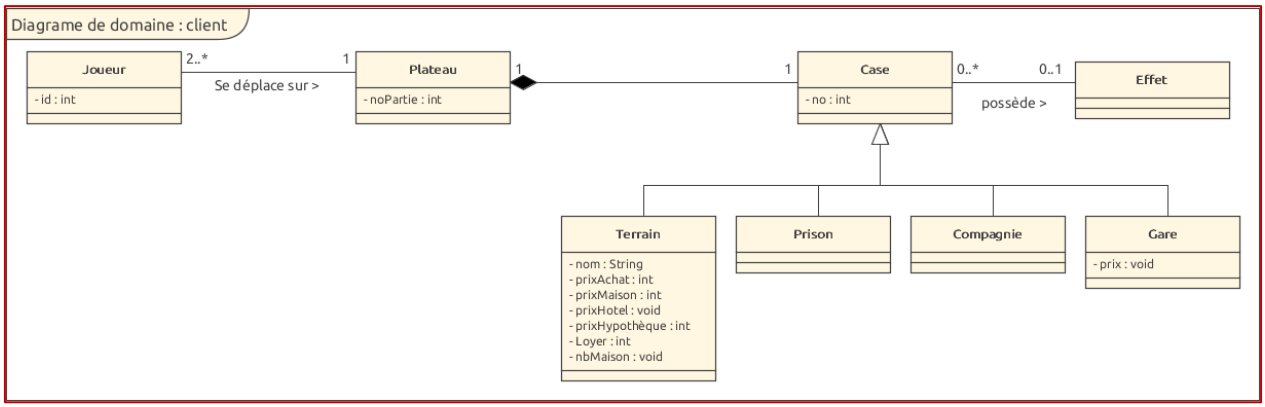


Figure 6 - Ebauche du modèle de domaine client

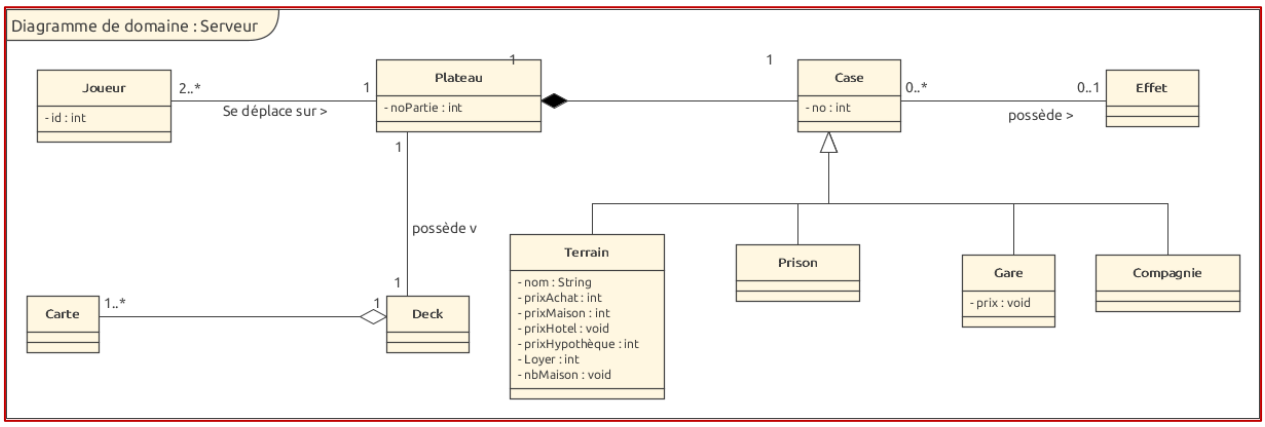


Figure 7 - Ebauche du modèle de domaine serveur

Voici ci-dessus, les différentes classes à implémenter côté client et serveur. Les classes sont les mêmes des deux côtés. La différence réside dans le fait que seul le serveur doit générer des objets de type Deck et Carte provenant de la base de données, afin de pouvoir choisir aléatoirement une carte.

Base de données

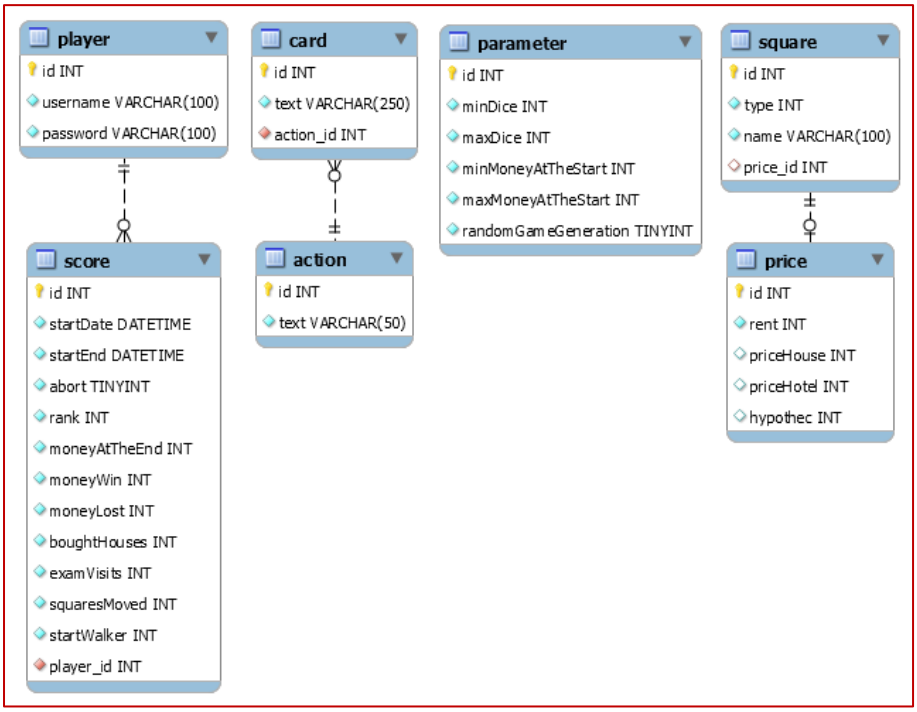


Figure 8 - Base de données de Cheseaux-poly

La base de données de notre projet est plutôt simpliste, car nous n'avons pas besoin de stocker d'autres informations que celles relatives aux joueurs, aux cartes, aux paramètres de jeu et aux cases.

Pour un joueur, nous gardons ses informations de connexion ainsi que les différents scores réalisés durant ses parties jouées. Un score est créé en début de partie et mis à jour en fin de partie, pour ne pas avoir d'interactions inutiles avec la base de données.

Toutes les cartes chance du jeu sont stockées dans la base de données. Pour chacune d'elles, est associé une action de type énumération (AVANCER, RECULER, EXAMEN, etc.). Le chiffre associé à ces actions (exemple : avancer de 3 cases, payer 1000.-) est généré aléatoirement dans le code, afin de ne pas avoir des cartes à action fixe.

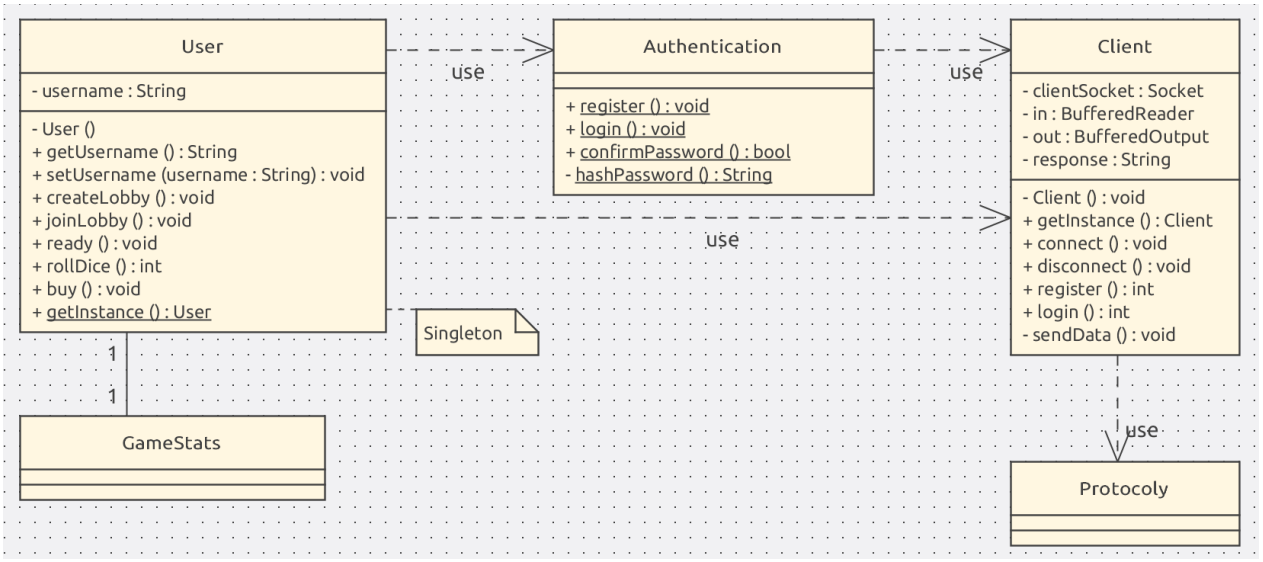
Un seul lot de paramètres est enregistré dans la base de données. Ce sont les limites générales du jeu fixées par l'administrateur. Ces limites sont fournies au joueur uniquement quand il créée une partie.

Chaque case du plateau de jeu est déterminée par un type (TERRAIN, EXAMEN, ENTREPRISE, etc.) et un nom. Celles nécessitant le paiement d'un loyer et/ou pouvant être achetées, possèdent des attributs financiers en plus pour permettre le paiement d'un(e) loyer/taxe, l'achat d'un(e) hôtel/maison et la mise en hypothèque.

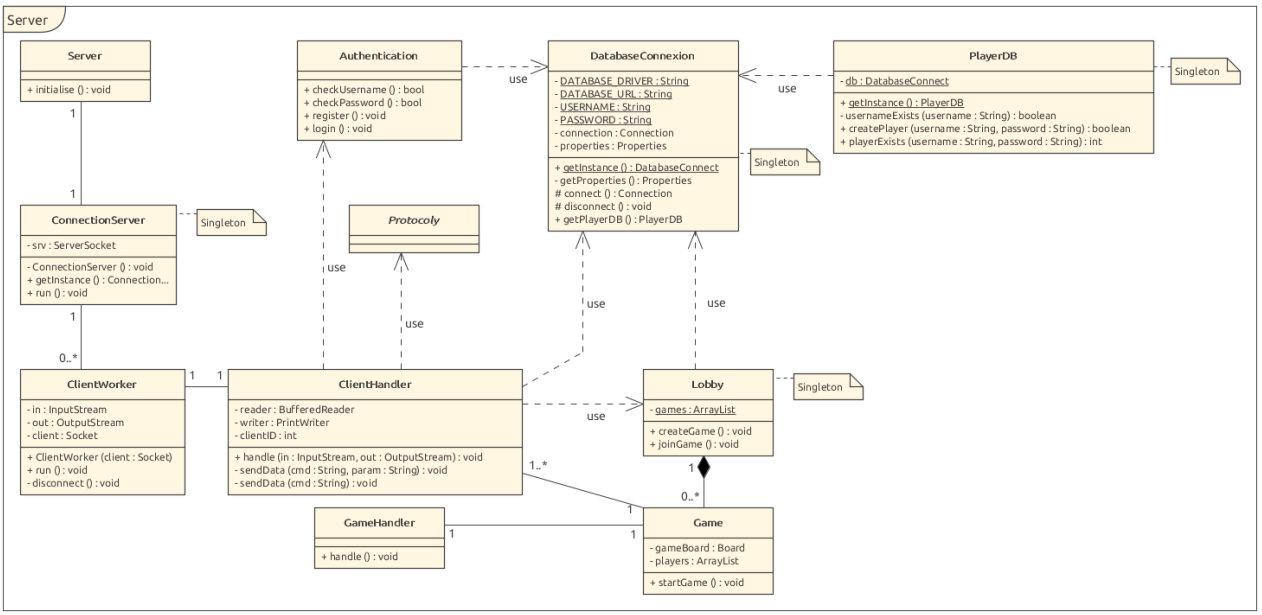
Concernant l'intégration de la base de données au sein de notre architecture, elle se situe du côté serveur. Ce dernier et le client s'échangent des messages protocolés. Les messages demandant une interaction avec la base de données sont générés et envoyés par le serveur uniquement. Le client n'a aucun lien direct avec la base de données. Il ne fait que transmettre une demande au serveur, qui analyse sa requête, génère une requête SQL associée, récupère la réponse, la formatte et la renvoie au client.

Schémas UML

Client



Server



Plan d'itérations

Sprint 1

BUT : Mise en place de l'enregistrement et de la connexion d'un joueur sur le jeu (base de données, serveur, client).

DURÉE : 19 heures (du 27 avril au 3 mai)

Figure 9 - Histoires du sprint 1

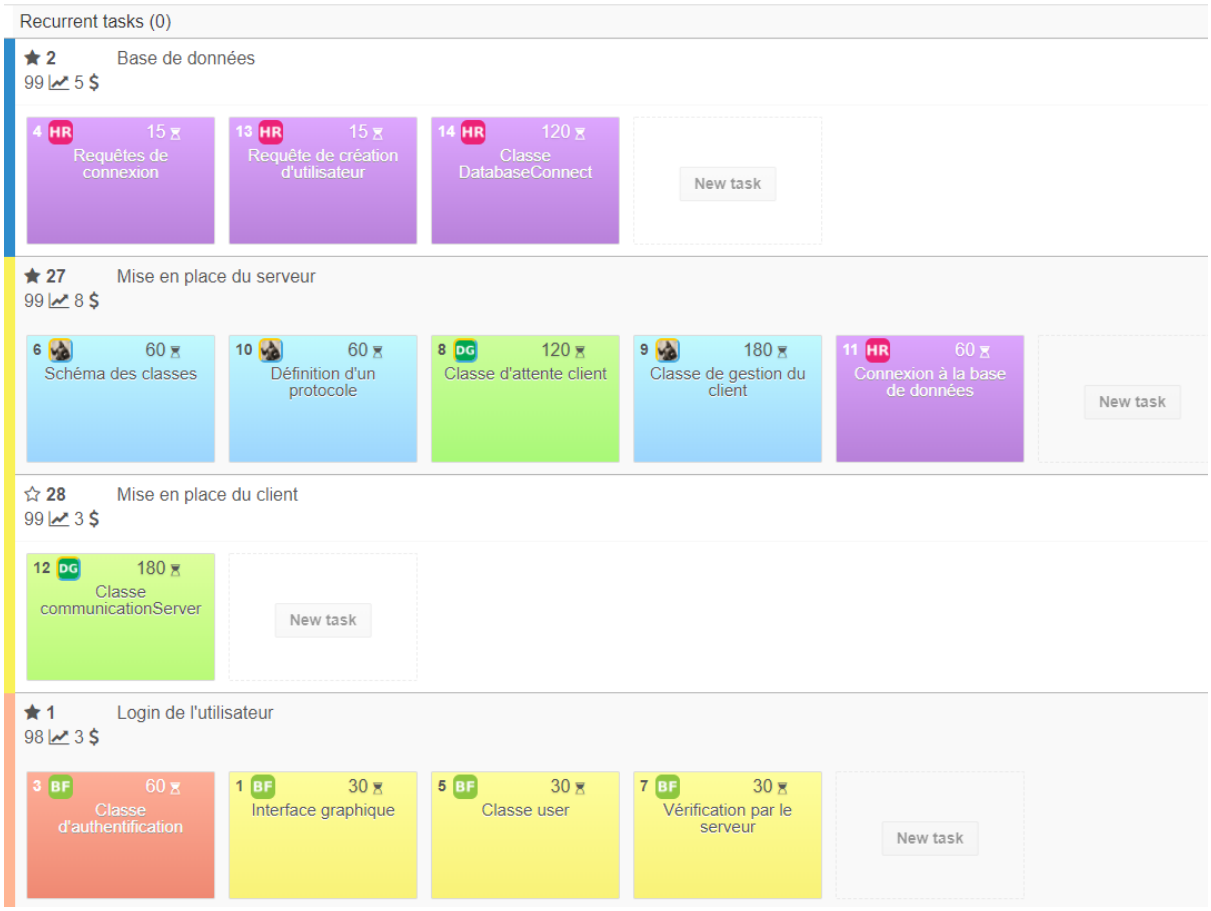


Figure 10 - Tâches du sprint 1

Sprint 2

BUT : Pouvoir, après s'être connecté au jeu, voir le salon de jeu avec les parties disponibles, créer sa propre partie avec des paramètres personnalisés et rejoindre une partie.

DURÉE : 17 heures (du 4 mai au 10 mai)

Figure 11 - Histoires du sprint 2

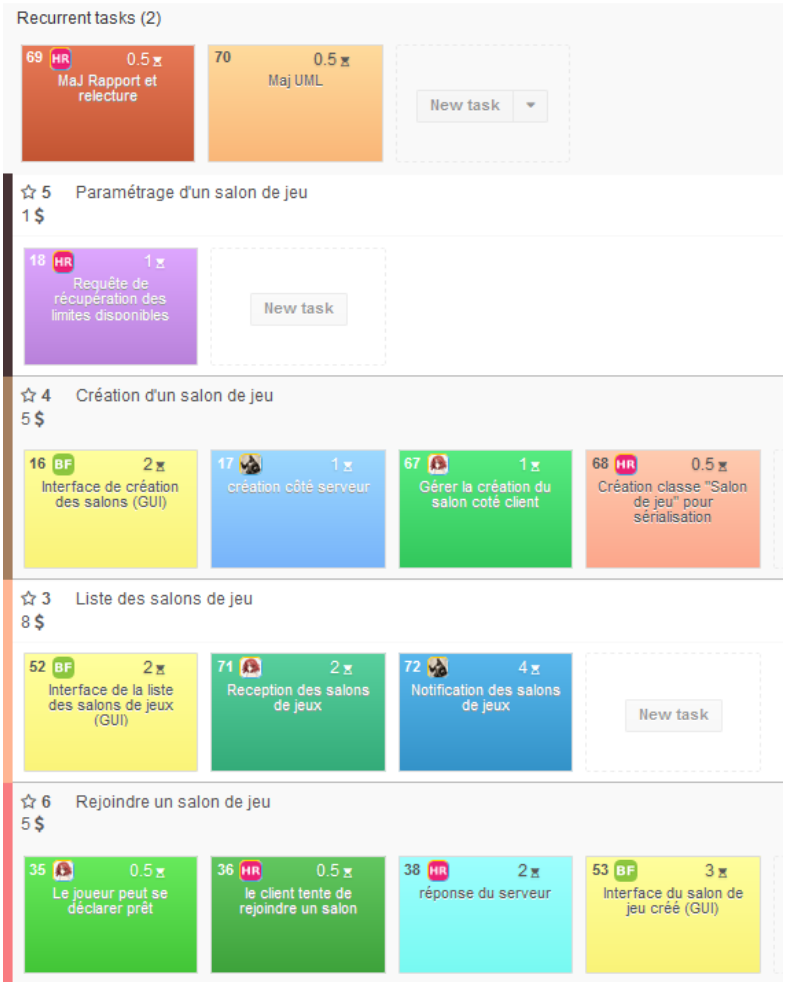


Figure 12 - Tâches du sprint 2

Bilans d'itérations

Vous trouverez ci-après les bilans des itérations rédigés par nos soins. Les bilans d'itérations effectués en compagnie du professeur et de l'assistant sont fournis dans les annexes.

Bilan 1

a) Bilan sur la terminaison des histoires

- Histoires planifiées : 1 – 2 – 27 – 28
- Histoires terminées : 1 – 2 – 27 – 28
- Histoires non-terminées : aucune

b) Vélocité du sprint

4 histoires réalisées/4 histoires planifiées → vélocité de 100%

c) Replanification

Le sprint ayant été complètement réalisé, aucune replanification n'est nécessaire.

d) Commentaire général

Tout s'est passé comme prévu et même plus rapidement que ce que nous pensions, avant d'entamer le sprint. Nous n'avons rencontré aucun problème durant le développement.

e) Autocritique

Le sprint s'est bien passé, néanmoins nous aurions pu faire une meilleure implémentation de certaines parties du code que nous allons devoir modifier de toute façon. Par exemple, nous aurions pu faire une meilleure gestion des erreurs, commenter le code de manière plus rigoureuse ou encore implémenter les messages d'erreurs lors de problèmes de connexion (interface graphique) plutôt que de faire des changements de couleur incompréhensibles pour un utilisateur lambda. Surtout que pour ce premier sprint, nous étions en avance et cela nous aurait fait gagner du temps pour la suite.

Mise à part ce petit point, tout le reste s'est bien passé, le sprint est concluant et opérationnel (tous les tests sont passés), nous n'avons donc pas eu d'adaptation à faire pour la suite et nous pouvons continuer sur cette bonne voie.

f) Bilans personnels

- Bryan (3.5 heures réalisées VS 4.5 heures planifiées)
  Une première itération qui s'est plutôt bien passée. L'implémentation d'un serveur est intéressante.
- Daniel (5 heures réalisées VS 5 heures planifiées)
  Fondamentalement, tout s'est bien passé. Le premier sprint était assez critique, car il met en place la connexion entre client et serveur (qui va être utilisée par la suite). Cela dit, nous avons bien implémenté cela, sans trop de problèmes, ce qui va nous permettre de bien poursuivre notre travail. L'effort fourni était conséquent, mais pas exagéré.
- François (1.5 heures réalisées VS 1.5 heures planifiées)
  Tout s'est bien passé et dans les temps.
- Héléna (2.5 heures réalisées VS 3.5 heures planifiées)
  Pour ma part, tout s'est bien passé. Je n'ai rencontré aucun problème au niveau de la base de données. Le point le plus délicat était de faire en sorte que mes collègues accèdent au serveur de base de données stocké sur mon ordinateur. Pour cela j'ai dû créer un utilisateur lié à la base

de données provenant de n'importe quelle adresse IP, et ajouter une règle dans mon pare-feu pour autoriser des connexions venant de l'extérieur sur le port de MySQL.

Bilan 2

g) Bilan sur la terminaison des histoires

- Histoires planifiées : x
- Histoires terminées : x
- Histoires non-terminées : x

h) Vélocité du sprint

x

i) Replanification

x

j) Commentaire général

x

k) Autocritique

x

l) Bilans personnels

- Bryan (x heures réalisées VS x heures planifiées)
  x
- Daniel (x heures réalisées VS x heures planifiées)
  x
- François (x heures réalisées VS x heures planifiées)
  x
- Héléna (x heures réalisées VS x heures planifiées)
  x

Annexes

Itération 1

No 1 - Login de l'utilisateur

en tant que Joueur
Je veux me connecter
afin de pouvoir rejoindre un salon

Démo1: Enregistrement
*Given*
L'utilisateur entre un nom d'utilisateur et un mot de passe pour s'enregistrer
*When*
Lorsque l'utilisateur clique sur le bouton
*Then*
☑ L'utilisateur arrive dans le menu des salons de jeu, si le nom d'utilisateur est utilisé, on affiche un message d'erreur

Observé:
Ca fonctionne, avec les messages d'erreur côté client et signalisation en console dans le serveur

Démo2: Connexion
*Given*
L'utilisateur entre un nom d'utilisateur et un mot de passe pour se connecter
*WHEN*
Lorsque l'utilisateur clique sur le bouton
*Then*
☑ L'utilisateur arrive dans le menu des salons de jeu, si le nom d'utilisateur et le mot de passe ne correspond pas, on affiche un message d'erreur

Observé:
Ca fonctionne, avec les messages d'erreur côté client et signalisation en console dans le serveur

No 28 - Mise en place du client

Démo1: Communication avec le serveur
☑ Le client doit être capable de se connecter au serveur,
☑ lui transmettre des commandes/messages,
☑ lire et interpréter les réponses de connexion

Observé:
Interface avec login et mot de passe et un bouton, le tout en couleur, dont le look change selon le mode enregistrement ou connexion.

Dès que le client est lancé, ce dernier se connecte au serveur.

No 2 - Base de données - Technical story

La base de données doit être crée, les tables initialisées, peut-être des données tests, requête pour créer les utilisateurs, et autres UPDATE

Démo1: Requête insertion utilisateur
☑ On teste manuellement la requête d'insertion d'un utilisateur.
☑ On vérifie qu'un utilisateur a été inséré dans la base de donnée

Observé:
Fonctionnel

Toute la base de données est déjà mise en place
Les tests "manuels" se font via le terminal.

Démo2: Requête de correspondance du mot de passe avec le nom d'utilisateur

- [x] On teste manuellement la requête qui vérifie qu'un mot de passe correspond à un nom d'utilisateur.
- [x] On regarde que cette requête nous retourne un seul ID.

Observé:
Fonctionnel

No 27 - Mise en place du serveur

Démo1: Démonstration

- [x] Le serveur doit répondre correctement aux réponse
- [x] il doit se connecter à la base de données
- [x] il doit récupérer les données de la base de données

Observé:
Les messages de connexion, d'enregistrement sont signalés en console

Démo2: Enregistrement manuel d'un utilisateur
Lancer le serveur
Ouvrir une fenêtre du terminal
Se connecter au serveur (telnet [address] [port])
Écrire la commande :
RGSTR [username] [password_digest]

- [x] Si username disponible, le serveur doit répondre :
  - [x] OK
  - [x] Aller voir dans la base de données si le nouvel utilisateur a été créé.
- [x] Sinon :
  - [x] DENIED

Observé:

Démo3: Connexion manuel d'un utilisateur
Lancer le serveur
Ouvrir une fenêtre du terminal
Se connecter au serveur (telnet [address] [port])
Écrire la commande :
LOGIN [username] [password_digest]

- [x] Si username existant et password correspondant, le serveur doit répondre :
  - [x] OK
- [x] Sinon :
  - [x] Si aucun username correspondant :
    - [x] UNKNOWN
  - [x] Si mot de passe erroné :
    - [x] DENIED

Observé:

Démo4: Vérifier le protocole

- [x] Tester toutes les commandes du protocole et vérifier les réponses envoyées par le serveur.

Observé:
Testé en console, message par message.

Remarque: un test unitaire aurait pu être mis en place.

Démo4: 2 requêtes de connexion en //
Ouvrir 2 terminaux
Connecter au serveur le premier terminal
- ☑ Tester d'envoyer une requête login par exemple

Connecter au serveur le deuxième terminal
- ☑ Vérifier si les deux terminaux sont gérés par le serveur en parallèle (en envoyant des commandes depuis les deux)

Observé:
Testé avec telnet alors qu'un autre client est connecté

Bilan général

Tout ok.
Des test unitaires ont été mis en place, notamment pour le client.
Base client-serveur "fait main" en se basant sur les labos RES.