Frogger

Tony Clavien, Maxime Guillod, Gabriel Luthier & Guillaume Milani

12 juin 2017

Table des matières

Table des figures

1 Introduction

Dans le cadre du cours GEN (Génie logiciel), nous allons créer une application client-serveur par groupe de 4 personnes. Nous avons choisi de développer un jeu reprenant le jeu *Frogger* qui consiste à faire traverser des routes pleines de trafic à une grenouille sans se faire écraser.

2 Fonctionnement général du jeu

2.1 Objectifs de base

Le but de ce projet est de réaliser une application client-serveur dans le cadre du cours GEN. Nous avons décidé de proposer un jeu de type « Frogger » en version valaisanne.

Le but du jeu sera de faire descendre une piste de ski à un Valaisan (*joueur skieur*). Il devra éviter les obstacles la traversant qui seront envoyé par le *défenseur*.

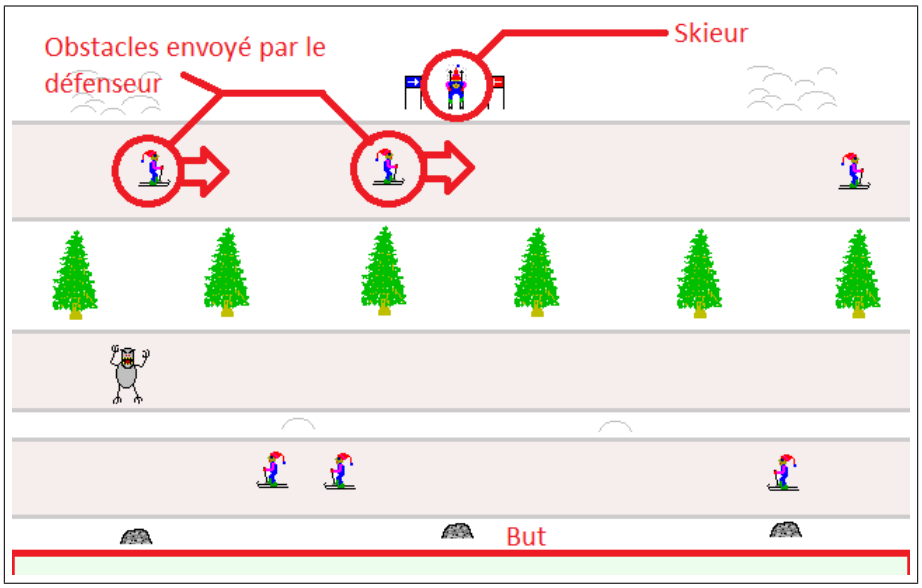

FIGURE 1 – Maquette d'une partie

Les zones rouges sont les zones à disposition du joueur 2 pour placer les obstacles, les sapins sont des obstacles fixes. Le joueur 1 gagne s'il atteint la zone en vert sans heurter d'obstacle.

2.2 Utilisation de l'applicatif

Le joueur 1 contrôle son personnage à l'aide des flèches du clavier (←↓→) et tente d'éviter les obstacles. Le joueur 2 place les obstacles dans les colonnes, ceci fait que l'obstacle se met en mouvement dans la colonne. Une bouteille de Fendant^{TM}se vide au fur et à mesure que le joueur 2 place des obstacles. Il doit ensuite attendre qu'elle se remplisse à nouveau pour placer de nouveaux obstacles.

L'administrateur peut régler les paramètres du jeu (« skin » du jeu, vitesse des obstacles, vitesse de rechargement de la bouteille de Fendant^{TM}). L'administrateur est un rôle hérité de celui de joueur (un administrateur est donc avant tout un joueur). Ce droit est stocké dans la base de données.

2.3 Règles du jeu

Pour le joueur 1 : le but est d'arriver en bas de la piste sans avoir heurté d'obstacle.
Pour le joueur 2 : le but est que le joueur 1 heurte un obstacle.

2.4 Contraintes

Le joueur 1 ne peut pas revenir en arrière, une fois descendu une portion de piste il peut s'arrêter, changer de «colonne» ou de continuer à descendre.

Le joueur 2 peut envoyer des obstacles pour autant que la bouteille de Fendant™ne soit pas vide.

2.5 Priorités de développement

1. Première version du jeu «standalone» avec un seul joueur qui prend le rôle du joueur 1 (descendre la piste). Les obstacles sont envoyés aléatoirement. (En parallèle, développement du serveur et de l'API).
2. Ajout du deuxième joueur en local (les deux jouent sur la même machine avec des touches différentes du clavier).
3. Les joueurs jouent chacun sur leur machine et se connectent à un serveur distant centralisant la partie. Lorsqu'un joueur se connecte, il voit la liste des parties crées (soit en attente d'un joueur, soit en cours). Il peut soit rejoindre une partie en attente d'un joueur soit créer une nouvelle partie.
4. L'administrateur peut configurer le serveur (paramètres des parties).

2.6 Base de données

La base de données stocke et partage les informations suivantes entre les joueurs et l'administrateur :

- Informations des joueurs (nom d'utilisateur, mot de passe, rôle, points totaux gagnés)
- Informations des parties (joueurs, points remportés par chacun)
- Paramètres de l'application (ce qui peut être configuré par l'administrateur)
- Logs de l'application

3 Analyse

3.1 Partage des responsabilités entre le serveur et le client

3.1.1 Responsabilités du client

Le client est en charge de transmettre les actions effectuée par le joueur au serveur. Ces actions sont en fait les touches appuyées qui correspondent à une action spécifique dans le jeu (comme par exemple aller à droite ou à gauche, ou envoyer un obstacle à un endroit spécifique). Le client doit aussi gérer l'affichage du jeu, en récupérant les emplacements de tous les éléments du jeu (skieur, obstacles fixes et obstacles dynamiques) à partir du serveur.

3.1.2 Responsabilités du serveur

Le serveur lui doit gérer la logique du jeu. Pour chaque action effectuée par un des joueurs, il va mettre à jour les emplacements des éléments du jeu et vérifie si la partie est gagnée (ou perdue) en regardant s'il y a eu une collision ou si le skieur est arrivé en bas de la piste de ski. Il fait ces mise-à-jours de manière asynchrone, mais il va contacter les clients de manières synchrone (toutes les x millisecondes) pour que les deux joueurs aient le jeu dans le même état chacun.

3.2 Diagramme d'activité général

La figure 2 représente le diagramme d'activité qui illustre la répartition des responsabilités entre le client et le serveur.

3.3 Rôle des participants

Joueurs : 2 personnes par partie, le *skieur* ayant pour but de réussir à descendre la piste, tandis que l'autre essaye de l'en empêcher (*défenseur*).

Administrateur : Peut changer les règles générales du jeu (configuration des difficulté, tailles des cartes, etc...), et s'occuper des données (nettoyer, modifier des joueurs, etc...)

3.4 Cas d'utilisation

3.4.1 Diagramme général de contexte

3.4.2 Description des acteurs

1. Joueurs, dans une partie nommés : *Skieur* et *Défenseur*
2. Administrateur

3.4.3 Scénario principal

1. L'un des deux joueurs se connecte au lobby et crée une partie en spécifiant les paramètres qu'il souhaite.
   (a) Difficulté
   (b) Taille de la carte
   (c) Son rôle (*Skieur* ou *Défenseur*)
2. Le second joueur rejoint la partie libre du joueur 1 dans le lobby et prends le rôle restant.
3. Lorsque les 2 joueurs sont prêts la partie commence.
   (a) "*Le skieur*" tente de traverser la piste en déplaçant son personnage de haut en bas.
   (b) "*Le défenseur*" tente de l'en empêcher en envoyant des obstacles de gauche à droite.
   (c) La partie se termine si le skieur n'a plus de vie ou si il a atteint l'autre côté (le but).
4. À la fin de la partie, soit les rôles s'inversent et on recommence une partie à l'étape 3, soit les joueurs quittent la partie. Cette dernière est donc interrompue et les joueurs retournent au lobby.

3.4.4 Extensions (ou scénarios alternatifs)

Pour permettre la récupération des parties de manière correct, il faut s'assurer que tous les états et les événements sensibles du système peuvent être récupérés à n'importe quelle étape du scénario.

Lors d'une partie, si l'un des deux joueurs est déconnecté, l'adversaire gagne automatiquement la partie et est renvoyé au lobby.

3.4.5 Scénario d'administration

1. l'administrateur se connecte à l'interface.
2. Il modifie l'une des options à sa disposition :
   (a) Les paramètres du jeu.
   (b) Les données d'un joueur.
3. puis il peut se déconnecte ou recommencer à l'étape 2.

3.5 Modèle de domaine

La figure 4 représente une ébauche du modèle de domaine.

3.5.1 Modèle de domaine pour le client

3.5.2 Modèle de domaine pour le serveur

3.6 Base de données

L'objectif de la base de données est principalement de stocker les paramètres du jeu et les données utilisateurs. Nous avons fait le choix de ne pas stocker d'informations concernant les parties en cours qui seront chargées uniquement dans la mémoire du serveur.

On trouve donc une entité `User` qui peut être `Administrator`, dont on stocke le login et le nombre de victoires. `Settings` modélise les réglages généraux du jeu, `MapSize` permettra à l'utilisateur qui créé une partie de choisir parmi plusieurs tailles de carte. `DifficultyLevel` modélise les niveaux de difficultés parmi lesquels l'utilisateur créant la partie aura le choix.

3.6.1 Modèle conceptuel (entité-associations)

4 Conception du projet

4.1 Protocole d'échange entre le client et le serveur

4.1.1 Communication

Nous allons utiliser JSON[1] comme protocole d'échange entre le client et le serveur.

1. JSON : JavaScript Object Notation

En effet, de part la structure de son contenu, l'échange d'information entre les deux intervenants est très facilement sérialisable/désérialisable, sans compter sur le fait que le contenu est lisible par un individu (contrairement à un contenu binaire).

4.1.2 Données échangées

L'API de communication en cours d'élaboration se trouve sur le wiki de notre projet GitHub
`https://github.com/gluthier/GEN-projet/wiki/Communication-API`
L'idée est que chaque joueur envoie au serveur ses commandes : pour le skieur s'il souhaite tourner à gauche ou à droite, pour le valaisan sur quelle ligne il envoie un obstacle. Le serveur s'occupe de placer les obstacles et le skieur et envoie les positions de manière régulière aux deux clients. Les clients ne font que l'affichage et l'envoi de commande alors que le serveur s'occupe de la synchronisation, gestion des collisions etc.

4.2 Diagrammes de classes du serveur et du client

4.3 Modèle conceptuel & relationnel de la base de données

5 Implémentation du projet

5.1 Structure du projet

Pour partager les différentes parties de l'applicatif, nous avons séparé le code en plusieurs projets afin d'exposer que le code qui est nécessaire. Le client, par exemple, n'a pas besoin d'avoir accès au code qui gère la base de donnée ni à code de l'applicatif administrateur.

5.1.1 GEN-protocol

`GEN-protocol` définit la protocol utilisé par le client et le serveur pour communiquer entre eux.

5.1.2 GEN-BDD

`GEN-BDD` offre une interface pour communiquer avec la base de donnée du jeu.

5.1.3 GEN-server

`GEN-server` est la partie que gère la logique du jeu et la synchronisation des joueurs entre eux.

5.1.4 GEN-admin

`GEN-admin` est l'applicatif administrateur qui permet de gérer la base de donnée, ajouter des comptes utilisateurs, consulter les logs en offrant une interface graphique.

5.1.5 GEN-Frogger

`GEN-Frogger` est le code destiné au client qui gère l'affichage graphique du jeu, ainsi que les interactions du joueur.

5.2 Technologies, langages, bibliothèques utilisés

Ce projet a été réalisé entièrement en Java, que ce soit le client ou le serveur. Nous avons utilisé JavaFX pour gérer la partie graphique du jeu. Il s'agit d'une bibliothèque de création d'interface graphique pour le langage Java. Elle contient notamment des outils pour du graphisme 2D que nous avons utilisés pour la gestion des éléments affiché à l'écran.

5.3 Technologies "originales" (autres que les technologies étudiées à l'école)

1. Descriptif
2. Avantages
3. Limitations
4. Remarques personnelles

5.4 Problèmes éventuels rencontrés et solutions apportées

6 Gestion du projet

6.1 Rôle des participants au sein du groupe de développement

1. Tony Clavien : Analyste, design manager, programmeur, représentant valaisan
2. Maxime Guillod : Chef de projet, programmeur
3. Gabriel Luthier : Responsable des tests, programmeur
4. Guillaume Milani : Architecte, concepteur en chef, programmeur, community manager

6.2 Plan d'itérations initial

6.2.1 Création de l'interface de base

Objectif : Mise en place de l'interface utilisateur de l'écran principal du jeu.

Durée : 1 semaine

Date du rendu : 26.04.2017

Partage des tâches : Création des images/sprites, affichage de l'UI, mise en place des obstacles fixes

Effort : 15 heures

6.2.2 Ajout du 1er joueur avec les obstacles

Objectif : Développement de la logique du joueur "skieur" qui descend la piste.

Durée : 1 semaine

Date du rendu : 03.05.2017

Partage des tâches : interaction avec les touches du clavier, déplacement du skieur sur la carte

Effort : 20 heures

6.2.3 Ajout du 2ème joueur en local

Objectif : Développement de la logique du joueur "défendant" qui envoie des obstacles contre l'autre joueur.

Durée : 1 semaine

Date du rendu : 10.05.2017

Partage des tâches : interaction avec les touches du clavier, logique de déplacement des obstacles sur la carte

Effort : 20 heures

6.2.4 Mise en place des logiques de jeu

Objectif : Détection des collisions entre les obstacles (fixes et mobiles) avec le joueur "skieur", des conditions de victoire et défaite et barre de recharge pour la pose d'obstacles.

Durée : 1 semaine

Date du rendu : 17.05.2017

Partage des tâches : détection de collisions, conditions de victoire, conditions de défaite, barre de recharge pour la pose d'obstacles

Effort : 20 heures

6.2.5 Ajout de la communication avec le serveur

Objectif : Développement du serveur pour pouvoir jouer à distance.

Durée : 1 semaine

Date du rendu : 24.05.2017

Partage des tâches : connexion des deux joueurs à une partie, communication réseau entre le serveur et les deux joueurs, gestion de l'état d'une partie

Effort : 30 heures

6.2.6 Application administrateur

Objectif : Développement de l'application administrateur utilisée afin de modifier les paramètres des différents modes de jeu et des données des joueurs.

Durée : 1 semaine

Date du rendu : 31.05.2017

Partage des tâches : UI de l'applicatif, connexion à la base de données, fonctions de modification de la BD

Effort : 20 heures

6.2.7 Ajout du Lobby

Objectif : Gestion du lobby (UI et logique).

Durée : 1 semaine

Date du rendu : 07.06.2017

Partage des tâches : UI du lobby (menu), récupérer les informations sur les parties en recherche de joueurs, création d'une nouvelle partie

Effort : 20 heures

6.2.8 Finalisation

Objectif : Finalisation de l'application et ajout de bonus.

Durée : 1 semaine

Date du rendu : 14.07.2017

Partage des tâches : derniers détails, mise en places des bonus/easter eggs

Effort : 15 heures

6.3 Suivi du projet

Itérations par itérations (bilan, problèmes rencontrés, replanifications, ...), synthèse Trello (sur une page environ)

6.4 Stratégie de tests

1. Effectués quand, par qui
2. Outils utilisés ?
3. Utilisation de JUnit pour au moins une classe conséquente
4. Résultats des tests

6.5 Stratégie d'intégration du code de chaque participant

Afin de collaborer efficacement, nous avons utilisé Git (avec la plateforme Github) pour gérer l'intégration du code de chaque développeur. Pour chaque fonctionnalité, une branche était crée pour permettre à différentes personnes de collaborer dessus. Puis, quand la fonctionnalité était finie, une pull request était ouverte sur Github pour permettre aux autres membres du groupe d'analyser et critiquer (si besoin) les changements effectué. Une fois tout le monde satisfait, la pull request était mergée sur la branche master qui correspond à l'ensemble de fonctionnalités finies.

7 État des lieux

7.1 Ce qui fonctionne (résultats des tests)

7.2 Ce qu'il resterait à développer (en proposant une planification)

8 Auto-critique

1. Relativement à votre solution technique, votre gestion de projet, votre plan d'itération
2. Ce que vous auriez pu améliorer et comment

9 Conclusion

A Manuel d'utilisation

1. Installation
2. Utilisation (avec copies d'écran)

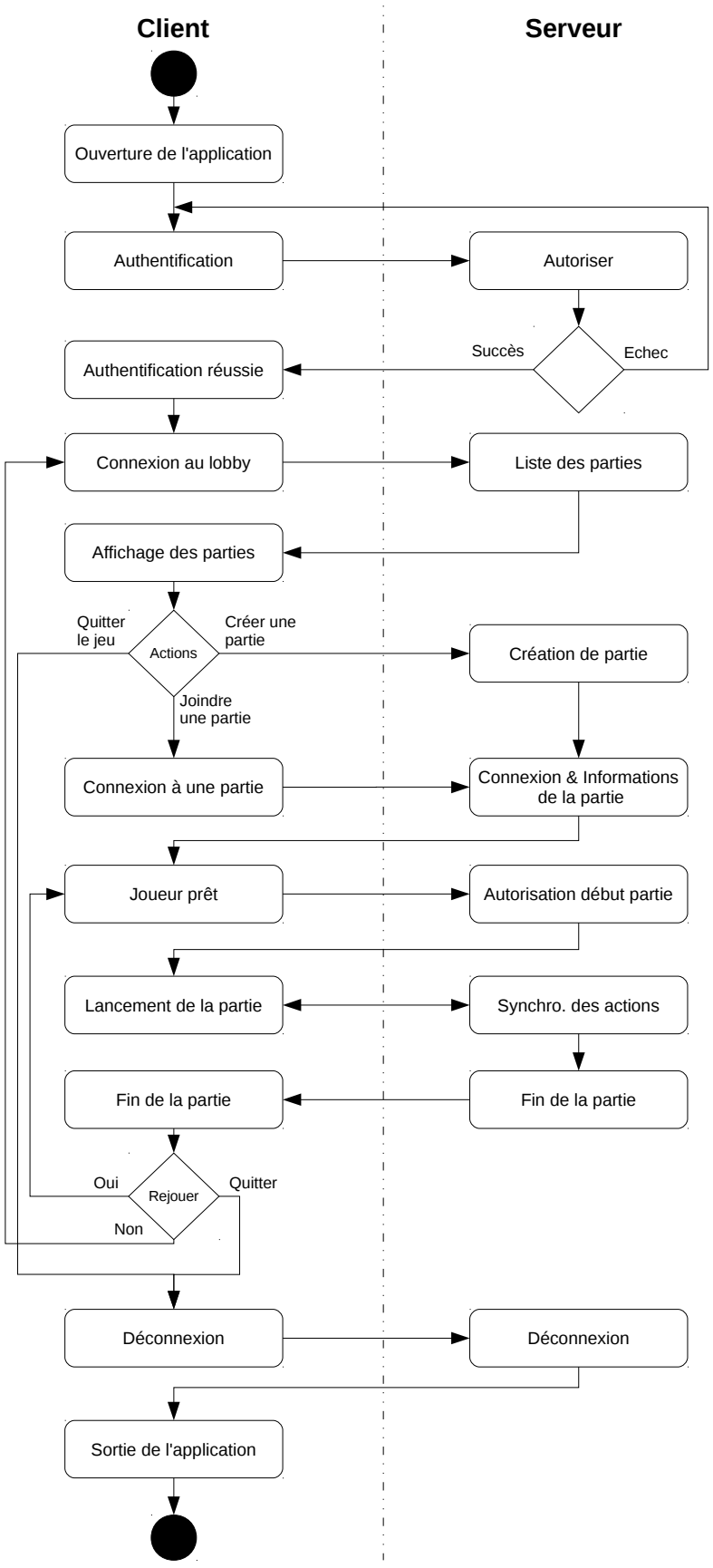

FIGURE 2 – Diagramme d'activité

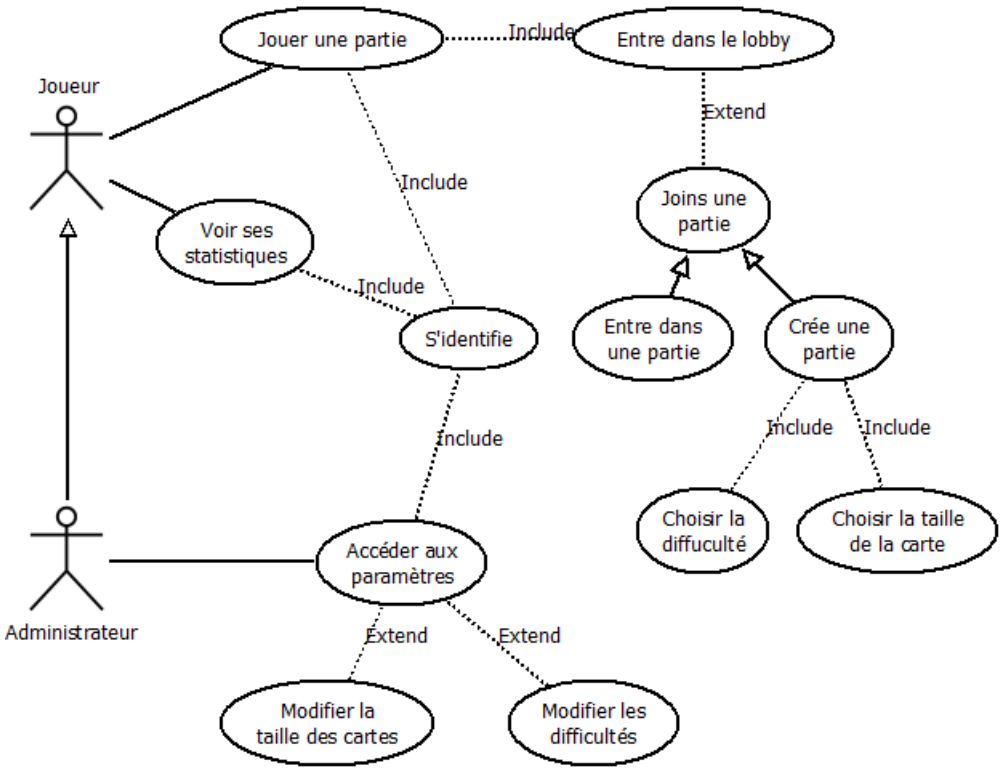

FIGURE 3 – Diagramme d'utilisation

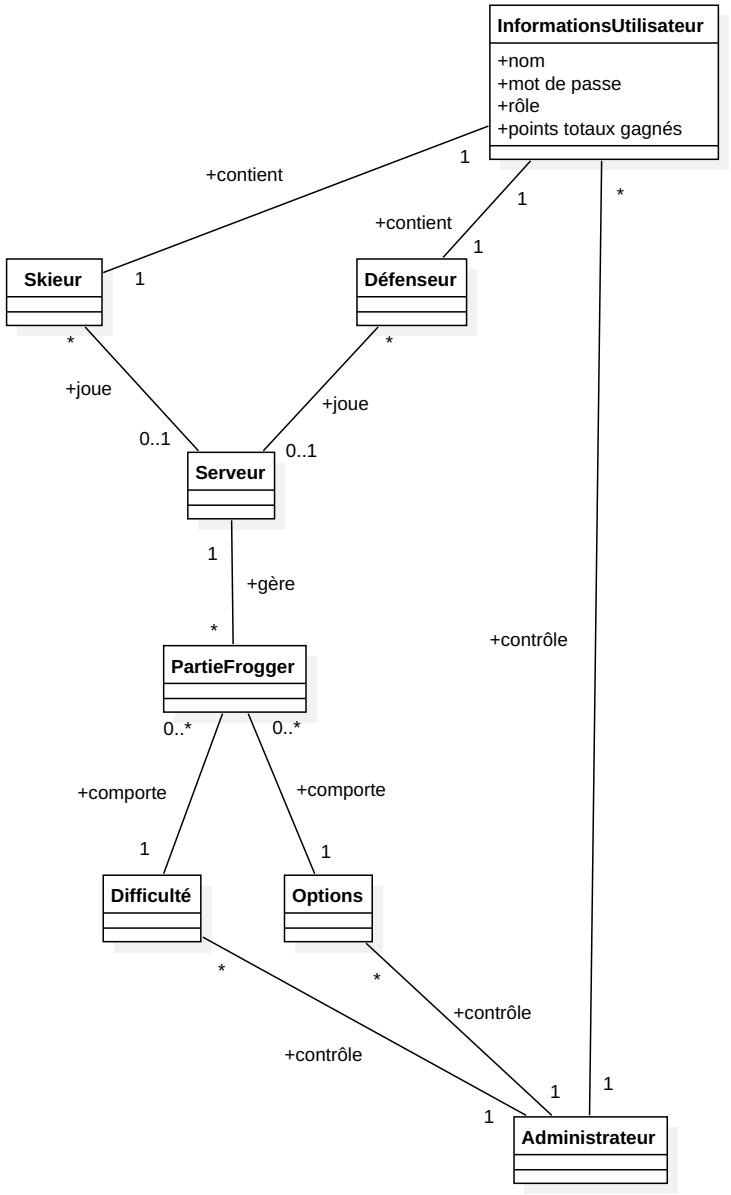

FIGURE 4 – Modèle de domaine

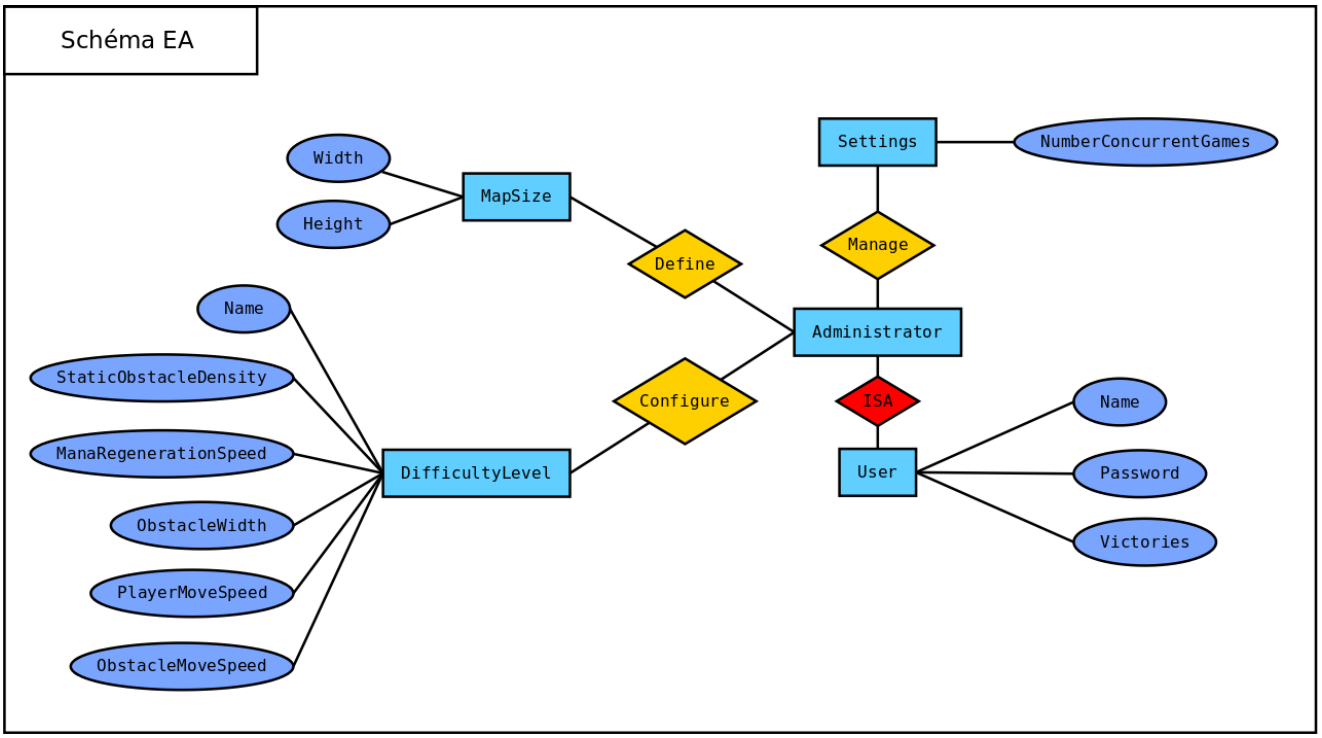

FIGURE 5 – Modèle conceptuel