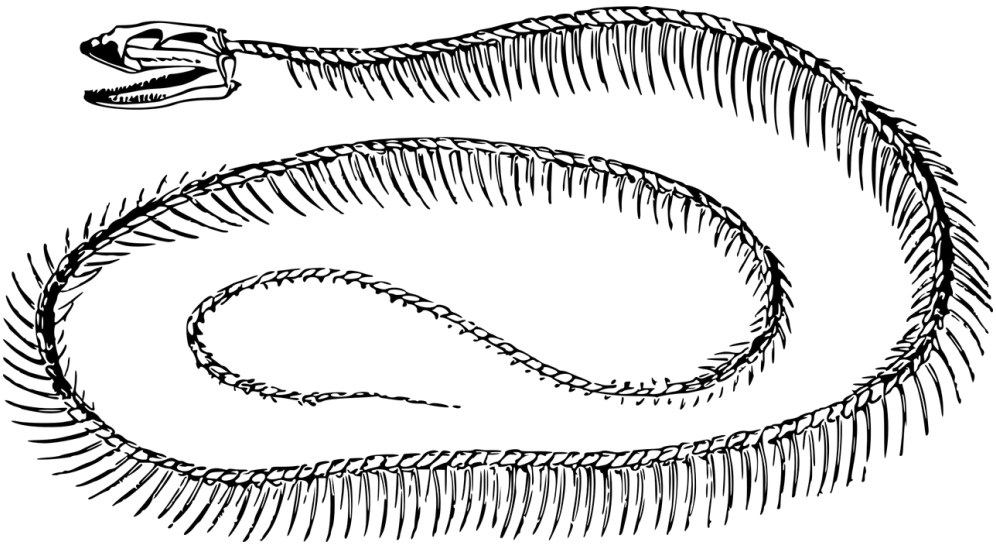
Master i informatique

Design Pattern

Décembre 2017



Projet de Développement

Snake en réseau

Auteurs

David Dembele

Alassane Diop

Rahmatou Walet Mohamedoun

Professeur

Benoit Da Mota

université angers

Table des matières

Chapitre 1

Introduction

1.1 Contexte

Dans le cadre de l'UE Design Pattern, de notre premier semestre de Master 1 Informatique à l'Université d'Angers, nous avons poursuivi la phase d'analyse d'un développement de jeu de Snake en réseau.

Le jeu peut se jouer en local ou en réseau, entre différents joueurs ce qui nécessite une implémentation des protocoles réseaux, pour gérer les transmissions entre les joueurs mais aussi entre les joueurs et le serveur.
Ceci nous a amené à mettre en place une sorte de document RFC [1] pour expliquer les protocoles réseaux utilisés et comment les utiliser.

La conception de ce jeu est assez complexe, raison pour laquelle il nous a fallu réfléchir sur le choix des technologies et pour cela nous avons préféré adopter un patron de conception [2] pour garantir une solution éprouvée et validée par des experts, mais aussi trouver une solution avec les bonnes pratiques de conception.

1.2 Objectifs

L'objectif attendu est de permettre à différents joueurs de se connecter au jeu et de jouer entre eux. Chaque joueur doit disposer d'un serpent qu'il contrôle à l'aide du clavier, avec les touches de direction (HAUT, BAS, GAUCHE, DROITE). Le serpent doit se déplacer pour ramasser des bonus ; ces bonus lui permettent de grandir, éventuellement de devenir momentanément invisible, d'augmenter ses points de vie, de faire un retour dans le temps pendant un certains nombre de secondes.
Les serpents pourront s'entre-tuer ; un serpent tué se transforme en bonus en faveur du serpent tueur ainsi le dernier serpent à survivre sera le gagnant.

Le jeu doit permettre aux joueurs de faire des demandes de connexion et de déconnexion, d'envoyer leurs déplacements au serveur.
Le serveur doit gérer les connexions et déconnexions des joueurs et centraliser les actions des joueurs. En effet il gère le plan de jeu partagé par tous les joueurs.
À chaque connexion d'un joueur, le serveur doit récupérer les informations du joueur, lui attribuer un serpent, son niveau de compétence et son niveau de jeu.
À chaque déplacement d'un serpent, il doit faire une mise à jour du plan de jeu pour prendre en compte la nouvelle position du serpent, en informer les autres joueurs.

1. Les requests for comments (RFC), littéralement « demande de commentaires », sont une série numérotée de documents officiels décrivant les aspects techniques d'Internet, ou de différents matériels informatiques (routeurs, serveur DHCP).Wikipédia

2. Un patron de conception (souvent appelé design pattern) est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels. Wikipédia

Chapitre 2

Protocole réseau

2.1 Motivation

Dans ce jeu, nous avons utilisé les protocoles TCP [1] et UDP [2].

Le protocole UDP nous sert dans le cas d'une transmission sans accusé de réception ; en effet le protocole UDP repose sur une communication unidirectionnelle. Lorsqu'une machine A envoie un paquet à une machine B, les paquets sont reçus par B sans accusé de réception vers la machine A.

Contrairement au protocole UDP, le protocole TCP repose sur une communication bidirectionnelle. La communication entre deux machines nécessite toujours un accusé de réception du récepteur vers l'émetteur. Dans le cas d'une communication où il faudra limiter la perte de paquet, TCP pourra nous servir.

2.2 Justification

Le protocole UDP est utilisé dans les cas suivants :

— déconnexion : lorsqu'un joueur souhaite quitter le jeu il envoie une requête de déconnexion au serveur, le serveur reçoit la requête et lui déconnecte du jeu sans lui en informer (sans accusé de réception).

— envoi du plan de jeu : lorsque le serveur centralise les actions des utilisateurs, il met à jour le plan du jeu et l'envoie aux joueurs ; cette opération peut être répétée autant de fois possible sans perturber le jeu.

Les informations destinées aux joueurs en provenance du serveur, seront envoyées via le broadcast [3].

Le protocole TCP est utilisé à chaque fois qu'il y ait échange entre le serveur et les joueurs sauf dans le cas d'une déconnexion ou d'un envoi du plan de jeu.

Les échanges d'informations entre le serveur et les joueurs se feront via des sockets et doivent être fiables, sans erreurs notamment dans le cas d'une connexion d'un joueur au serveur, de la réception par le serveur des actions des joueurs. La communication entre le serveur et les joueurs reposent sur des requêtes HTTP. Ci dessous une représentation de cette communication :

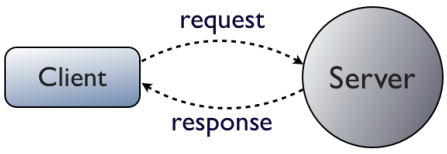


FIGURE 2.1 – Communication entre joueur (client) et le serveur

1. Transmission Control Protocol (littéralement, « protocole de contrôle de transmissions »), abrégé TCP, est un protocole de transport fiable, en mode connecté. Wikipédia
2. Le User Datagram Protocol (UDP, en français protocole de datagramme utilisateur) est un des principaux protocoles de télécommunication utilisés par Internet. Il fait partie de la couche transport du modèle OSI, il appartient à la couche 4, comme TCP. Wikipédia
3. Le broadcast permet au serveur d'envoyer un paquet à tous le joueurs connectés

2.3 Éventuels problèmes

Assurer les communications entre les joueurs et le serveur, en essayant de garantir l'intégrité des paquets, s'avère complexe ; des erreurs liées à la transmission des paquets peuvent survenir.
En effet, à la fin d'une partie, il peut y avoir un paquet renseignant la position ou le déplacement d'un serpent ; si ce paquet arrive après la réception du paquet indiquant la fin de la partie, le joueur émettant le paquet pourrait être bloqué.

2.4 Implémentation

2.4.1 Le jeu

Ci dessous les étapes du déroulement du jeu :

joueur1	–>	serveur	Connexion au serveur
joueur1	–>	serveur	Création d'une nouvelle partie
serveur	–>	joueurs	Notification nouvelle partie
joueurs	–>	serveur	Participer à la partie
serveur	–>	joueurs	Envoie du plan de jeu
joueurs	–>	serveur	Réception du plan de jeu

... Les joueurs commencent à joueur ...

joueurX	–>	serveur	Participer au jeu
serveur	–>	joueurX	Participation acceptée
serveur	–>	joueurs	Notification nouveau joueur
joueurY	–>	serveur	Remporte la partie
serveur	–>	joueurs	Notification partie remportée
serveur	–>	joueurs	Fin de la partie

Au cour du jeu, chaque joueur envoie ses déplacements au serveur qui les centralise et notifie à tous les joueurs qu'ils doivent rafraîchir leur plan de jeu.

2.4.2 Les commandes

Les échanges entre le serveur et les joueurs reposent sur la norme ISO-8859 [4] qui, a l'avantage d'être connue par tous les systèmes d'exploitation et ne requiert que 8 bits par caractère.
8 bits = 1 byte

Les commandes des joueurs commenceront par le caractère '/' suivi du nom de la commande.
Exemple : */quit*
Cette commande permet à un joueur de quitter une partie de jeu.

Toutes les réponses du serveur seront représentées comme suit :
— *1xx* réponse de succès d'une commande
— *2xx* réponse d'erreur d'une commande
— *3xx* Commande du serveur

4. ISO 8859, également appelée plus formellement ISO/CEI 8859, est une norme commune de l'ISO et de la CEI de codage de caractères sur 8 bits pour le traitement informatique du texte. Wikipédia

Pour simplifier le transfert de données et éviter des conflits avec le XML (voir 3), les caractères suivants ne seront pas acceptés :

— '"' guillemets
— '<' signe d'infériorité
— '>' signe de supériorité

Le caractère ' ' (espace) est autorisé, s'il s'agit de séparer des blocs de données. Une commande ou un paramètre ne peut contenir de caractère ' ' (espace).

Toutes les commandes devront terminer par le caractère \n pour marquer la fin de la commande.

2.4.3 Messages de succès

Ci dessous, les messages de succès renvoyés par le serveur :

100 "Message de bienvenue"	message d'acceptation global
101 "pseudo" ok	pseudo accepté
102 "couleur" "xml"	Le joueur de couleur "couleur" rejoint le jeu ; "xml" (voir 3) représente ses données
103 "couleur" parti	Le joueur a quitté le jeu
104 changement niveau	Changement du niveau de jeu dans le serveur
300 debut partie	Commande pour notifier le début de la partie à tous les joueurs
301 fin "couleur"	Fin de la partie. *"couleur"* désigne le joueur qui remporte la partie.
302 nbJoueurs X couleur	Commande pour envoyer le plan de jeu à tous les joueurs
303 affichage plan	Commande pour notifier à tous les joueurs d'afficher le plan de jeu

2.4.4 Messages d'erreur

Ci dessous, les messages d'erreur renvoyés par le serveur :

200 "commande" inconnue	Le serveur ne connaît pas la commande
201 format invalide	Le format de la commande n'est pas valide
202 caractère invalide	Le caractère en question n'est pas accepté (voir 2.4.2)
203 "pseudo" invalide	Le pseudo est déjà utilisé
204 identification echouee	Le joueur n'a pas encore choisi de pseudo
205 "pseudo" deja	Le joueur a déjà un pseudo
206 "direction" invalide	La direction n'est pas valide (voir 6)
207 "couleur" invalide	La couleur est déjà attribuée à un autre joueur

2.4.5 Les commandes du serveur

— *300 debut partie*	Commande pour notifier le début de la partie à tous les joueurs
— *301 fin "couleur"*	Fin de la partie. *"couleur"* désigne le joueur qui remporte la partie.
— *302 nbJoueurs Xcouleur*	Commande pour envoyer le plan de jeu à tous les joueurs
— *303 affichage plan*	Commande pour notifier à tous les joueurs d'afficher le plan de jeu

2.4.6 Protocole UDP

Le serveur peut être lancé en mode Unicast ou Multicast.
En mode Multicast, le serveur envoie de manière automatique les informations, ainsi les joueurs pourront écouter sur le port prévu à cet effet.
En mode Unicast, chaque joueur peut demander des informations au serveur qui réponds immédiatement.

2.4.7 Protocole TCP

2.4.7.1 Joueur vers Serveur

1. Messages

100 "Message de bienvenue"	Message reçu par le joueur une fois connecté au jeu.
200 "commande" inconnue	Lorsque le serveur reçoit une commande qu'il ne connaît pas.
201 format invalide	Lorsque le format de la commande n'est pas valide.
204 identification echouee	Lorsqu'un client non connecté essaie d'envoyer un message.

2. Nom du joueur
   /pseudo "pseudojoueur"
   Cette commande permet de définir le pseudo du joueur. Cette commande doit être la première à exécuter par le joueur. À l'exception de la commande */quit*, toutes les autres commandes sont ignorées jusqu'à ce que le joueur définit son pseudo.

   Ci dessous les réponses possibles à cette commande :
   — *101 "pseudojoueur" ok*
   — *203 "pseudojoueur" invalide*
   — *205 "pseudojoueur" deja*

3. Participer au jeu
   /join
   Cette commande nécessite permet de participer au jeu.

   Ci dessous les réponses possibles à cette commande :
   — *100 "Bienvenue au jeu"*
   — *204 "identification echouee"*
   Une couleur est attribuée au joueur. Une liste XML lui est envoyée, voir exemple ci-dessous :

   102 bleu <joueurs>
   <joueur pseudo="pseudo" couleur="couleur" score="score" />
   ...
   </joueurs>
   À chaque fois qu'un joueur rejoigne le jeu, un message est envoyé aux autres joueurs déjà présent dans le jeu, pour leur notifier de sa présence. *102 "couleur" "xml"*

4. Quitter le jeu
   /quit
   Cette commande permet au joueur de quitter le jeu.

   À chaque fois qu'un joueur quitte le jeu, un message est adressé aux autres joueurs :

   103 "couleur" parti
   Informer aux autres joueurs restants que le joueur de couleur *"couleur"* a quitté le jeu.

5. Notifier la réception du plan de jeu
   /receptionPlan
   Cette commande notifie au serveur la réception du plan de jeu par un joueur.

   La commande */receptionPlan* ne retourne pas de messages.

6. Déplacer le serpent
   /turn H | B | G | D

G	aller à	gauche
D	aller à	droite
H	aller en	haut
B	aller en	bas

   Ci dessous les réponses possibles à cette commande :
   — *206 "direction" invalide*
   — *100 "direction valide"*

7. Fin de la partie
   /finPartie "couleur"

   "couleur" correspond à la couleur du joueur qui remporte la partie.

   Ci dessous la réponse possible à cette commande :
   100 "Fin de la partie"

2.4.7.2 Serveur vers Joueurs

Les messages renvoyés par le serveur notifient un changement d'état ; ils ne nécessitent pas de confirmation excepté au début du jeu (voir 3).

1. Début de la partie
   300 debut partie

   Notifier à tous les joueurs que le jeu a commencé.

2. Fin de la partie
   301 fin "couleur"
   Notifier à tous les joueurs que la partie est terminée. *"couleur"* désigne la couleur du joueur qui remporte la partie.

3. Envoi du plan de jeu
   302 nbJoueurs couleur1 ... couleurN
   — *nbJoueurs* nombre de joueurs participant au jeu
   — *couleur1 ... couleurN* les couleurs des joueurs participants

   Avant le début de chaque partie, le plan de jeu est envoyé à tous les joueurs. La partie commencera une fois que tous les joueurs auront notifié au serveur la réception du plan de jeu.

4. Affichage du plan de jeu
   303 affichage plan

   Une fois que le serveur aura reçu toutes les notifications de réception du plan de jeu, il ordonne aux joueurs d'afficher le plan de jeu afin qu'ils puissent savoir la position de chaque concurrent.

5. Nouveau joueur

   102 "couleur" "xml"

   — *"couleur"* couleur attribuée au nouveau joueur
   — *"xml"* données du nouveau joueur

   Lorsqu'un nouveau joueur rejoint le jeu, une notification de sa présence est envoyée à tous les autres joueurs déjà présents dans le jeu.

6. Déconnexion d'un joueur

   103 "couleur" parti

   — *"couleur"* couleur du joueur parti

   Notifier aux autres joueurs la déconnexion au jeu du joueur de couleur *"couleur"*.

Chapitre 3

Conception

3.1 Cas d'utilisation

Les échanges entre les joueurs et le serveur peuvent être résumés par les cas d'utilisation suivants où les acteurs sont les joueurs :

— Se Connecter
— Entrer son nom
— Deconnecter
— Creer une partie de jeu
— Initialiser le jeu
— Jouer
— Consulter l'aide
— Consulter score
— Inviter joueur
— Quitter le jeu

Ci dessous le diagramme des cas d'utilisation :

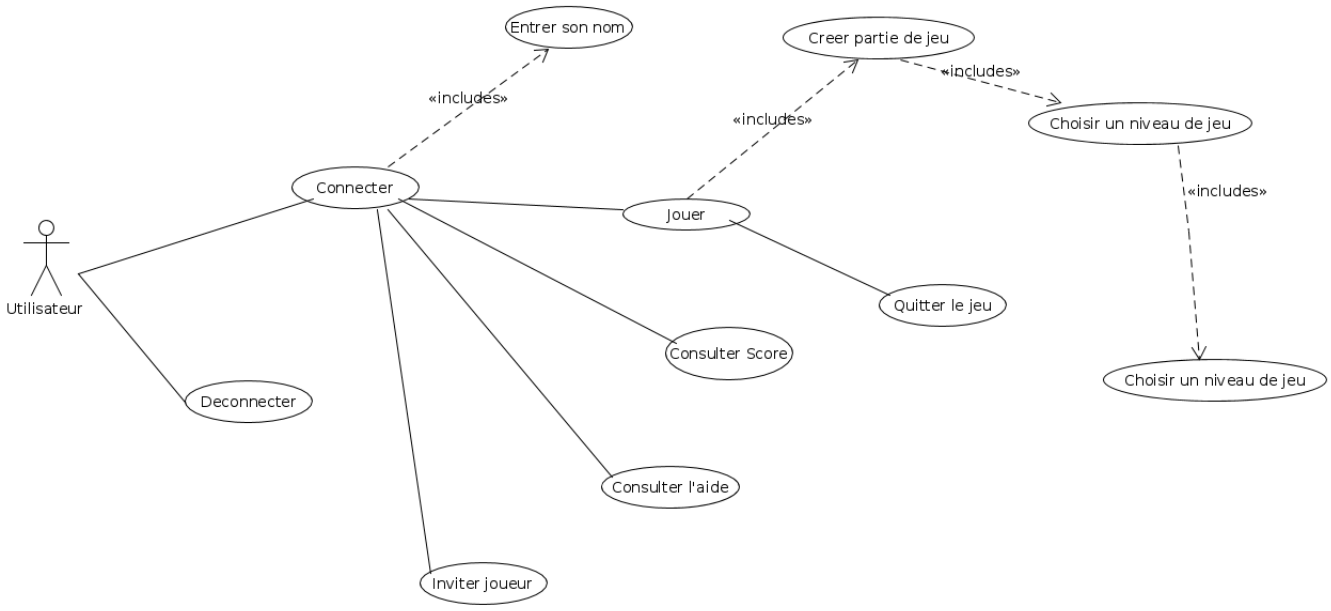


FIGURE 3.1 – Diagramme des cas d'utilisation

3.2 Points de changement et d'évolution

Durant une partie de jeu, beaucoup d'objets sont amenés à être modifiés. En effet, les actions effectuées par les joueurs entraînent d'états. Ainsi pour un joueur donné, nous pouvons dire que les points de changement concerne :

- le serpent : le comportement du serpent change ; il se déplace, ramasse des bonus, s'allonge au fur et à mesure du jeu. Sa visibilité peut changer ainsi que son nombre de vie.
- le niveau de jeu : à chaque fin de partie, les joueurs peuvent changer de niveau ; ils passent d'un niveau inférieur à un niveau supérieur.
- le plan de jeu : il change en fonction du niveau de jeu. Chaque niveau de jeu propose un plan de jeu différent, plus complexe.
- le nombre de joueurs : au début du jeu, il y a un nombre de joueurs défini ; ce nombre est amené à évoluer positivement si un joueur rejoint la partie et négativement si le serpent d'un joueur meurt.
- le score : plus un joueur ramasse des bonus, plus son score augmente.
- l'accès au serveur : l'accès au serveur d'un joueur peut passer d'un état connecté à un état déconnecté et inversement.
- une partie de jeu : elle peut passer d'un état début à un état en cours, d'un état en cours à un état pause, d'un état en cours à un état terminé.

3.3 Design envisagé

Notre application repose sur une architecture 3-Tiers [1]

Cette architecture nous permet de mieux distinguer chaque partie, d'où une conception plus optimale. Ainsi, nous aurons trois parties distinctes :

- couche client (les joueurs) : cette partie regroupe les joueurs qui se connectent au serveur. Nous aurons une abstraction des joueurs physiques, une représentation des joueurs en mémoire du côté de la couche client. Ci dessous le diagramme de classe de la couche client :

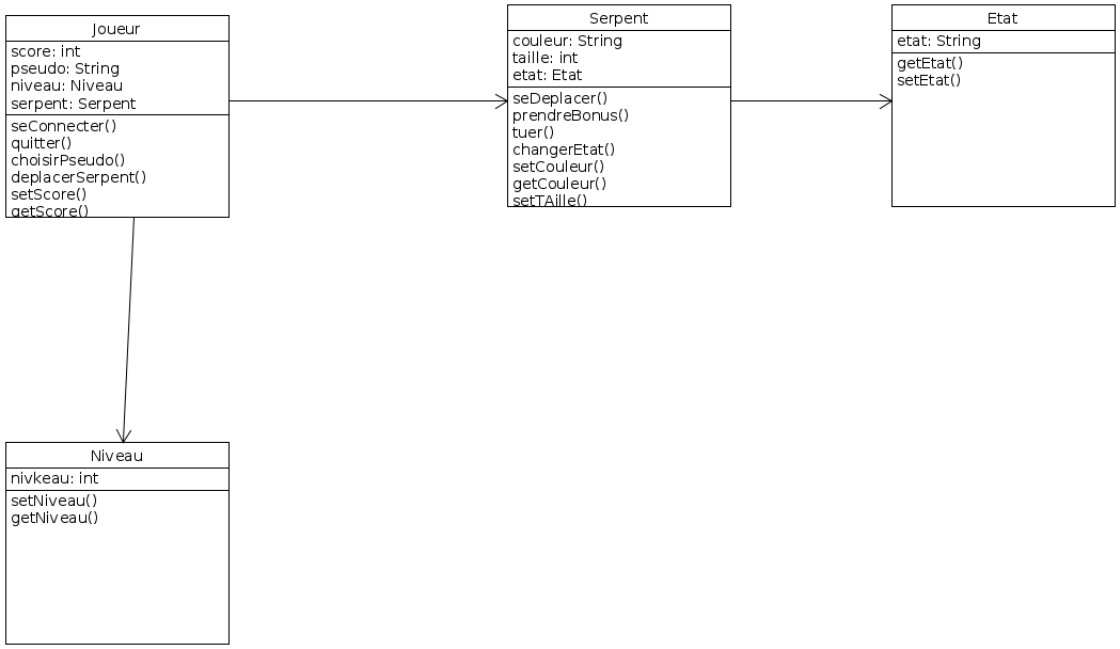


FIGURE 3.2 – Diagramme de classe de la couche client

1. L'architecture trois tiers, aussi appelée architecture à trois niveaux ou architecture à trois couches, est l'application du modèle plus général qu'est le multi-tier. L'architecture logique du système est divisée en trois niveaux ou couches : couche de présentations, couche de traitement, couche d'accès aux données. C'est une architecture basée sur l'environnement client-serveur. Wikipédia.

— couche serveur : cette partie regroupe à la fois le serveur, la vue, le modèle (les classes métiers).
Le serveur assure tout ce qui est communication avec les joueurs, centralise leurs actions. Le modèle est une abstraction des joueurs physiques, une représentation des joueurs en mémoire du côté dans la couche serveur. La vue représente le plan, la partie visible qui sera partagée par tous les joueurs, la partie qui permet aux joueurs d'interagir avec le jeu, serveur.

Afin de mieux séparer les composants de cette couche, nous structurons cette couche en utilisant le patter MVC, ainsi nous aurons de manière distincte le serveur qui est en même temps le contrôleur, la vue et le modèle. Ci dessous une représentation du pattern MVC :

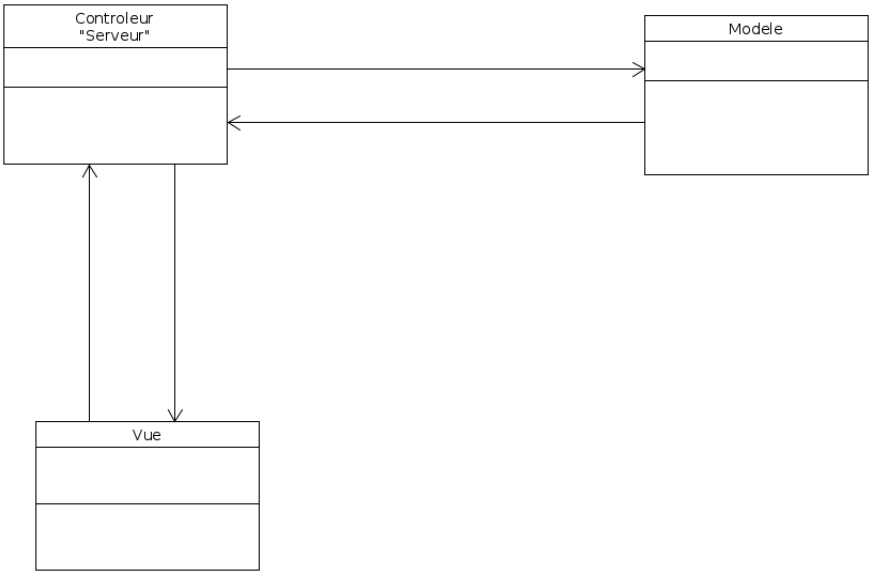


FIGURE 3.3 – Représentation du design pattern MVC

Le serveur communique avec la vue et le modèle via le pattern Façade. Ci dessus, le diagramme des classe du pattern MVC intégrant le pattern Façade du serveur :

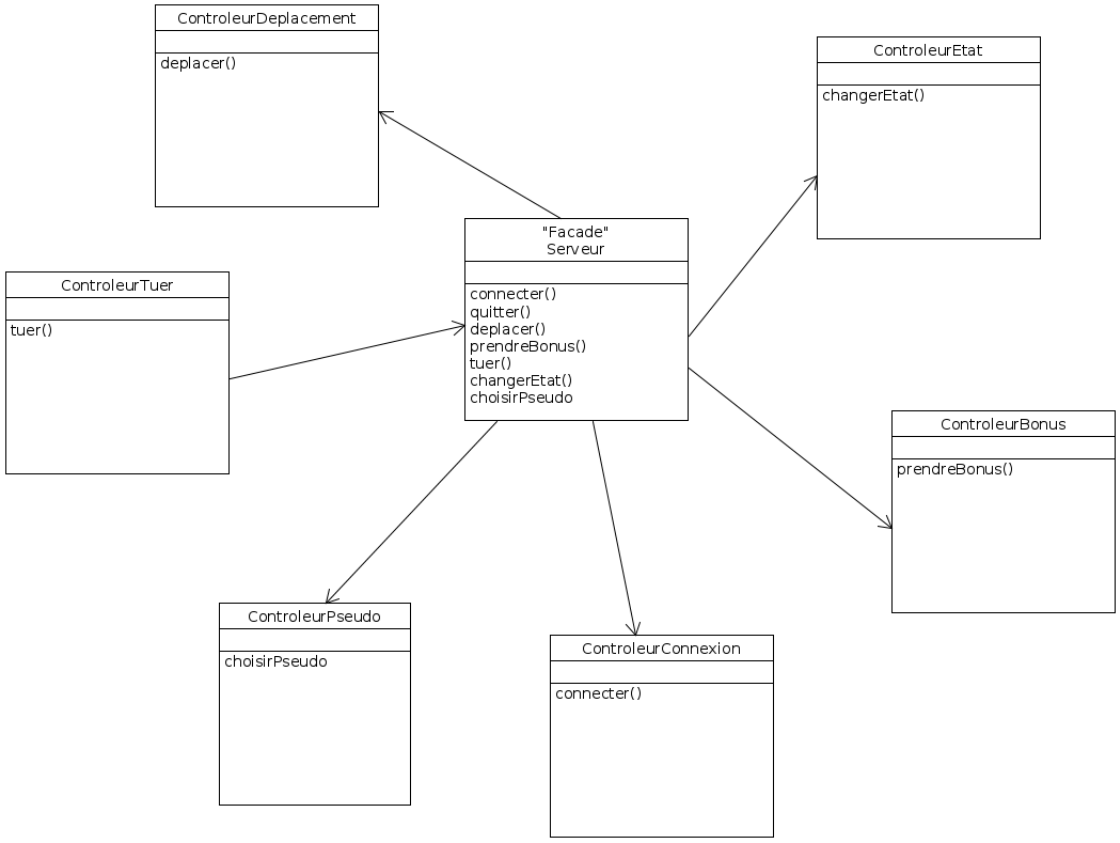


FIGURE 3.4 – Diagramme de classe de la couche serveur

— couche persistance : la persistance des données est gérée dans cette partie.
Les données sont une abstraction des joueurs en base de données. Dans cette couche, les données sont des objets des classes métiers.

Pour faire communiquer ces trois couches, nous mettons en place une conception basée sur différents designs patterns.

La communication entre la couche client et celle serveur se fera via le pattern Proxy. Ci dessus une représentation de la communication les couches client et serveur :

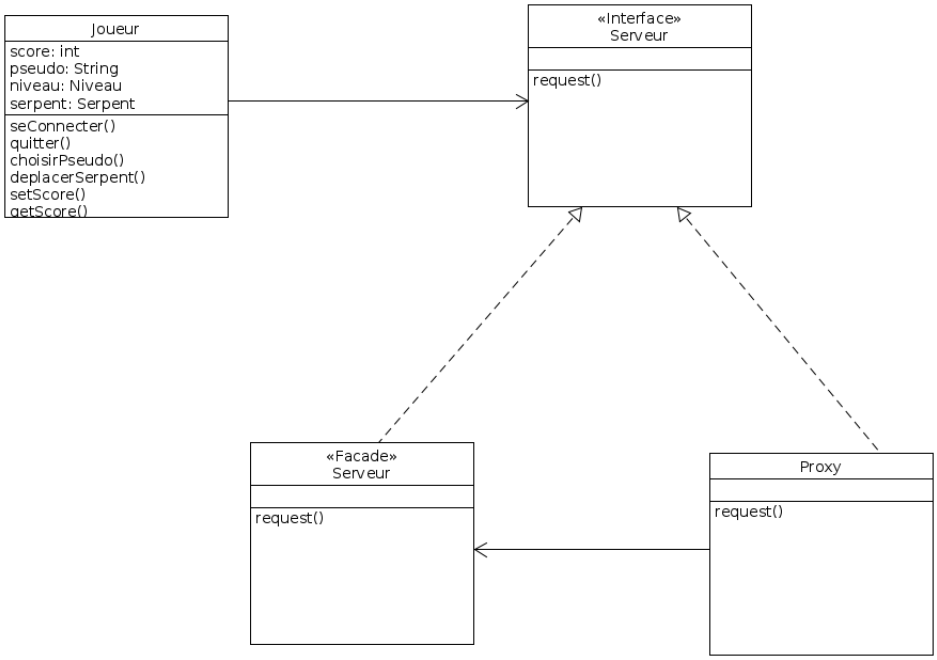


FIGURE 3.5 – Diagramme de classe du pattern Proxy

La persistance des données passe par le pattern DAO. Ci dessous une représentation de la persistance des données :

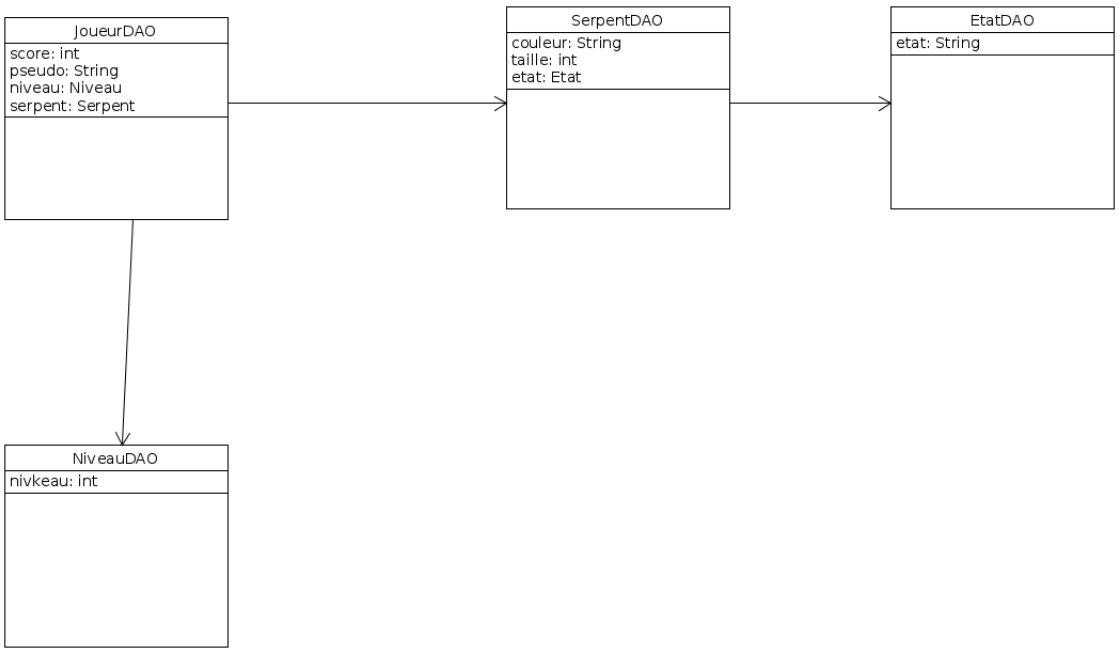


FIGURE 3.6 – Diagramme de classe du pattern DAO

3.4 Justification

Nos choix sur les patterns envisagés ne sont fortuits et sont motivés par le souci de faire une conception ouverte à l'évolution.

L'usage du pattern MVC pour la couche serveur, nous permet de dissocier les composants de cette couche, une séparation claire des responsabilités.

Étant donné la communication à distance entre le serveur et les joueurs, nous utilisons le pattern Proxy qui, est très utilisé pour la gestion d'objets distribués (protocole RMI en Java par exemple). L'idée étant de construire un Proxy capable de communiquer avec un objet distant (usage de la sérialisation en Java) sans que l'exploitant fasse de différences entre un accès local ou un accès distant.

En raison de la complexité des actions du serveur, nous avons jugé utile d'encapsuler ses actions en fournissant une interface unifiée à l'ensemble des sous systèmes qu'on peut avoir dans le serveur ; d'où l'usage du pattern Façade qui fournit une interface de plus haut niveau qui rend le sous système du serveur plus facile à utiliser.

Ainsi la couche client communique avec la couche serveur via l'interface unifiée fournie par le serveur, avec le pattern Proxy. La vue et le modèle communiquent également avec le serveur (contrôleur) via l'interface unifiée fournie par le serveur.

Pour assurer la persistance des données des joueurs de manière optimale, nous avons séparé la couche de persistance du reste de l'application ; ceci permet à notre application d'être indépendante de la couche de persistance et d'anticiper sur l'évolution de l'application.

Pour cela nous utilisons le design pattern DAO ; pour chaque classe du modèle, on aura une classe DAO correspondante. De ce fait toutes les opérations relatives à la persistance des données seront traitées dans les classes DAO.

Chapitre 4

Conclusion

Tout au long de cette conception, nous avons essayé de mettre en place des solutions souples et évolutives, qui pourront être facilement maintenables.

Cependant, des points restent à développer, à savoir donner aux joueurs la possibilité de créer autant de parties de jeu que possible, de pouvoir inviter d'autres joueurs à rejoindre une partie de jeu, de sauvegarder et restaurer une partie de jeu.