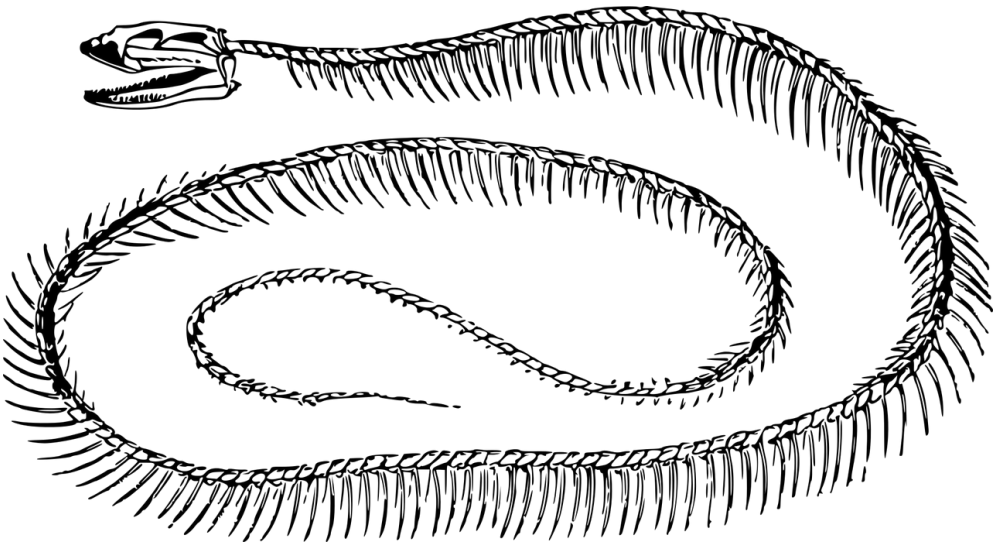
Master i informatique

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.

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 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.

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.

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

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 [3].

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.

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 peuvent contenir un caractère ' ' (espace).

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

3. 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

2.4.3 Messages acceptés

2.4.4 Messages d'erreur

2.4.5 Les commandes du serveur

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.
   — 208 $identification\ echoue$: 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 pseudo valide.
   — 203 pseudo déjà utilisé.
   — 210 le joueur a déjà un pseudo.

3. Participer au jeu
   $/join$ "$motDePasse$"
   Cette commande nécessite un mot de passe pour participer au jeu. Si aucun mot de passe n'est requis alors $motDePasse\ =\ *$

   Ci dessous les réponses possibles à cette commande :
   — 204 Mot de passe requis
   — 205 Mot de passe incorrect
   — 103 Bienvenue au jeu.
   Une couleur est attribuée au joueur. Une liste XML lui est envoyée, voir exemple ci-dessous :

   103 $bleu\ <joueurs>$
   $<joueur\ pseudo="pseudo"\ couleur="couleur"\ score="score"/>$
   ...
   $</joueur>$
   À chaque fois qu'un joueur rejoigne le jeu, un message est envoyé aux autres joueurs déjà présent dans le jeu.
   307 "$couleur$" "$pseudoJoueur$" $a\ rejoint\ le\ jeu$

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 :

212 "*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 :
   — 212 lorsqu'une direction inconnue est envoyée
   — 218 la direction est valide et prise en compte

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 :
   220 notification envoyée avec succès.

2.4.7.2 Serveur vers Joueurs

Les commandes du serveurs sont envoyés pour notifier un changement d'état ; elles ne nécessitent pas de confirmation excepté au début du jeu (voir 3).

1. Début de la partie
   302 *debut*
   Notifier à tous les joueurs que le jeu a commencé.

2. Fin de la partie
   303 *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
   300 *nbJoueurs couleur*1 ... *couleurN*

nbJoueurs	nombre de joueurs participant au jeu
*couleur*1 ... *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

   301 $affichage\ plan$

   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

   307 "*couleur*" "*pseudo*" *rejoint le jeu*

   "*couleur*" couleur attribuée au nouveau joueur
   "*pseudo*" pseudo 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

   212 "*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

3.2 Points de changement et d'évolution

3.3 Design envisagé

3.4 Justification

Chapitre 4

Conclusion