CAMUS Axel
ETCHEVERRY Nicolas
Groupe A4

Rapport Projet Réseau

Réalisation d'un Pong en réseau en java

L3 informatique 2015-2016

Présentation des fonctionnalités

Objectif : Réaliser le jeu Pong en réseau en évitant les possibilités de triche avec au moins 2 joueurs. Le jeu pourra disposer de balles spéciales et de bonus.

Objectifs atteints :

- Il est possible de jouer en réseau à deux joueurs sur des machines différentes
- Chaque client vérifie l'intégrité des données en calculant la position de tous les objets et en les comparant aux valeurs reçues des autres clients.
- Il est possible d'ajouter facilement des effets sur les objets : tous les objets héritent du même type qui a des attributs « speed » et « position ».
- On pourrait créer autant de balles que l'on veut, en choisissant leurs images.
- Un écran d'attente animé s'affiche lorsqu'on attend un autre joueur.

Objectifs non atteints :

- Pas de bonus
- Le multijoueur à plus de 4 joueurs n'est pas terminé, car nous n'avons pas eu le temps de faire le protocole de connexion, ni les raquettes horizontales, mais les fonctions d'envoi et de réception sur le réseau prennent en compte une liste de joueurs et envoient l'information à tous les joueurs à la fois.

Organisation

Nous sommes en binôme, Axel Camus et Nicolas Etcheverry, la majeure partie du développement a été faite à deux sur le même ordinateur, pour des raisons de réflexion et de coordination. Les commits effectués sur le git sont donc la plupart du temps communs.

Le projet est disponible sur GitHub à l'adresse suivante :

https://github.com/Etcheverry/pong_reseau

Travail réalisé

Dans cette section, nous présenterons et expliquerons nos choix d'implémentation. L'ordre n'est pas forcément chronologique.

Implémentation du jeu

PongItems

Dans un premier temps, nous avons créé une classe « PongItem » qui représente tous les objets pouvant se déplacer à l'écran. Les PongItem permettent de stocker l'image, la position et la vitesse des objets. Nous avons choisi de stocker tous les PongItem de la scène dans un ArrayList car c'est un moyen simple d'avoir un tableau ordonné sans avoir à gérer sa taille à chaque fois.

Les PongItems ont une méthode « animate » qui, dans notre première implémentation, met à jour la position de l'item par rapport à sa vitesse et à sa position actuelle. L'item est ensuite dessiné par « draw ». Les classes Ball et Racket héritent de PongItem.

Racket

Les objets « Racket » sont initialisés à l'aide d'un « id » qui détermine leur position sur le plan (gauche ou droite). Ils peuvent ensuite être pilotés en utilisant les fonctions « up », « down », « nup » et « ndown » qui permettent respectivement de monter, descendre, arrêter de monter, et arrêter de descendre. Nous avons décidé d'utiliser ces fonctions pour éviter les blocages de direction lorsqu'on appuie sur les 2 boutons à la fois. Dans les premières versions sans réseau, le pilotage de l'autre raquette se faisait avec les touches Z et S de la même façon.

Ball et Collisions

Au début, les collisions étaient gérées avant le déplacement de l'objet (deux objets étaient en collision s'ils se superposaient). Nous avons réalisé que cette méthode n'était pas efficace. En effet, si nous déplacions directement les objets, ceux-ci pouvaient se superposer ; ce qui pouvait entraîner des situations un peu singulières comme par exemple une balle restée coincée dans la raquette (nous parlons d'expérience).

Ainsi, nous avons choisi d'utiliser une valeur « nextpos » qui est calculée par « updateNextPos » afin de pouvoir prévenir les collisions avant d'afficher les objets superposés. Les collisions sont détectées uniquement par les balles dans la fonction « collision » de la classe Ball. La fonction « updateNextPos » est étendue pour appeler « collision » et modifie la trajectoire de la balle en cas de collision avec une raquette. La nouvelle trajectoire est calculée dans « speedAfterCollision » en fonction de la position de la balle sur la raquette.

Réseau

La partie réseau du programme se déroule en 2 étapes : la connexion des joueurs et la phase de jeu.

Classe Player

Tout d'abord, nous avons choisi de stocker toutes les informations relatives à chaque joueur auquel on est connecté dans la classe « Player ». Le joueur local, c'est à dire celui qui utilise la machine, n'est pas référencé dans les Player.

Un Player possède plusieurs attributs :

- Un « id » unique qui représente son numéro de joueur (joueur 0, joueur 1, joueur 2…) .
- Une socket permettant de communiquer avec lui

Et méthodes :

- « connection » permet d'ouvrir une connexion avec une autre adresse
- « write » permet d'envoyer un message au Player
- « read » permet de lire ce que le Player envoie
- « closeConnexion » ferme la connexion

Connexion des joueurs

La classe Network contient les fonctions permettant d'établir une connexion entre les joueurs.

Lors du lancement du programme, l'utilisateur peut soit lancer l'exécutable sans paramètre et devenir l'hôte, soit lancer l'exécutable en précisant une adresse et devenir un client.

L'hôte lancera la fonction « waitForNPlayers » avec en paramètre le nombre de joueurs à attendre et créera un « Player » pour chaque connexion entrante jusqu'à avoir assez de joueurs. Ces players sont stockés dans un tableau « players » et numérotés avec un « id ». À chaque fois, il enverra au joueur le nombre total de joueurs et son « id ». Son « id » personnel sera 0 et sera stocké dans l'attribut « id » de la classe « Network ».

En théorie, pour jouer à plus de 2, il faudrait ensuite envoyer à chaque joueur les adresses et les « id » des autres joueurs pour qu'ils se connectent entre eux, mais nous n'avons pas implémenté cette partie.

Le client quant à lui crée un Player pour l'hôte auquel il assignera l'id 0. Il s'y connecte et reçoit le nombre total de joueurs et son id. Il crée ensuite le tableau « players » dans lequel il range tous les joueurs.

Ici aussi, le client devrait recevoir les adresses des autres joueurs pour le multijoueur à plus de 2. Cette partie n'est pas implémentée, nous ne rangeons donc que le Player de l'hôte dans le tableau.

Ainsi, chaque machine dispose d'un tableau players contenant tous les Player, même s'ils sont dans un ordre différent, et est connectée à tous les autres joueurs.

Phase de jeu

Toutes les 10ms, la fonction « animate » de la classe Pong est appelée. Celle-ci mettra à jour pour chaque élément la position des balles puis enverra leur vitesse et leur prochaine position à tous les autres joueurs via les methodes « sendPosition » et « sendSpeed » de la classe Network. Les données reçues sont ensuite analysées par la fonction « analyze » (Nous décrivons leurs fonctionnement dans la partie « Analyse »). Pour les raquettes, on n'envoie que la prochaine position de notre raquette, les autres seront mises à jour dans « analyze ». On appelle ensuite la fonction animate sur tous les items pour mettre leur position à jour.

Protocole

Le protocole utilisé est très simple : le message est une chaîne de caractères découpée par des espaces. Le premier élément indique une opération, les suivants dépendent de l'opération effectuée.

Par exemple, un message de position est découpé ainsi :

- « pos <id> x y » = « L'item <id> est en position (x, y) »
- « pos 0 0 300 » = « L'item 0 est en (0, 300) »

La classe Network dispose de fonctions permettant de construire facilement de tels messages et de les récupérer sous forme de tableau : [pos, 0, 0, 300] .

Analyse

La fonction « analyze » de la classe Pong traite les informations données dans le tableau vu précédemment. Elle différencie les opérations et vérifie la cohérence entre les données reçues des autres joueurs et les données calculées pour les items. Si les résultats sont différents au-delà d'un certain seuil, on considère que les machines sont trop désynchronisées et le jeu est quitté. Si les résultats sont différents mais en dessous du seuil, on considère la moyenne de toutes les valeurs obtenues et on la stocke dans « nextpos » de l'objet. La même opération est effectuée pour vérifier la vitesse.

Pour les raquettes, le fonctionnement est un peu différent : on déplacera la raquette de « saved_speed » dans la même direction que la position proposée par le joueur. Ceci permet de l'empêcher de proposer un déplacement de raquette trop élevé.

En cas de déconnexion d'un joueur, le jeu de l'autre est fermé.

Sécurité

Malgré les vérifications de la méthode analyse, il est possible avec un client modifié de tricher un peu : si l'on propose à chaque fois une valeur de vitesse supérieure sans dépasser le seuil, puisque tous les clients vont calculer une moyenne, on peut légèrement changer la vitesse de la balle. On peut limiter cet abus en abaissant le seuil mais ce seuil est nécessaire car la fonction de calcul de vitesses ne renvoie pas forcément la même valeur en fonction des machines (elle fait des divisions).

Il est impossible de tricher sur la position de la raquette car même en envoyant des valeurs erronées, on ne considère que le signe de la valeur reçue.

La sécurité vis à vis des valeurs des objets est assurée dans les limites du seuil.

Interface

Interface de Pong

L'affichage est fait via la fonction « updateScreen » située dans les classes « Pong » et « WaitingScreen ». Elle est appelée à chaque fin de la fonction « animate » pour dessiner les objets et le fond à leur nouvelle position.

Score

Le score est affiché à chaque itération de « animate » pour chaque joueur de son côté de la ligne centrale. Il est mis à jour à l'aide de la fonction « updateScore » qui l'augmente quand des points sont marqués.

Nous avions prévu une fonction « sendScore » dans Network qui n'est finalement pas utilisée. L'envoi du score aux autres joueurs n'est pas nécessaire tant qu'il n'a pas d'utilité (par exemple pour les bonus, s'ils arrivent en fonction du score, etc) et la modification du score dans un client n'affectera que son affichage, et pas les autres clients.

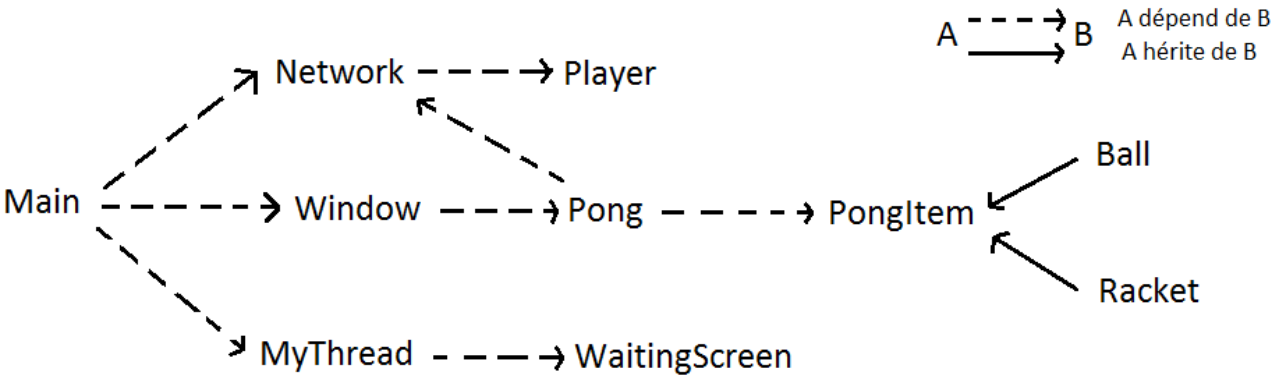
Écran d'attente

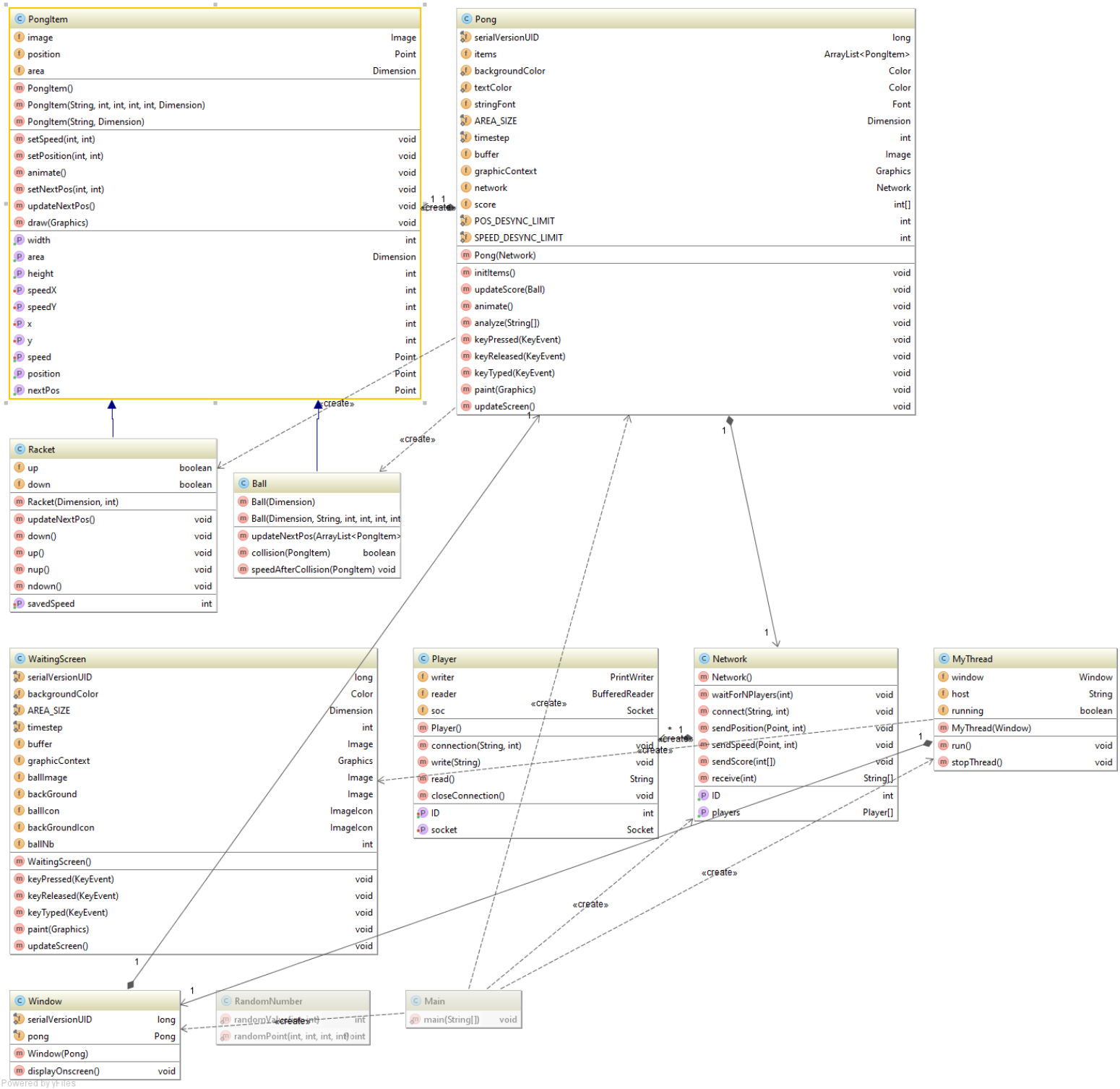
Nous avons ajouté un écran d'attente au début du programme pour accompagner l'attente ou la connexion des joueurs. Le code de cet écran est contenu dans la classe « WaitingScreen » qui reprend les éléments de base permettant un affichage similaire à celui de « Pong ».

Afin de le faire fonctionner en parallèle de l'attente de connexion, nous l'avons placé dans un thread. Notre première version plaçait les connexions dans le thread et l'écran en principal mais il est plus logique de faire le contraire car ici l'affichage est temporaire.

L'écran d'attente est stoppé lorsque la connexion est établie.

Diagramme des classes





Conclusion

Le projet de base est réussi : il est possible de jouer à Pong en multijoueur réseau. Cependant la plupart des objectifs annexes sont incomplets : les bonus ne sont pas disponibles mais facilement implémentables grâce à la forme de PongItem et au stockage des objets par identifiant. Il suffirait d'étendre la fonction « collision » de Ball et créer celle de Racket pour leur permettre de toucher des bonus.

De plus, le multijoueur à plus de 2 joueurs est partiellement implémenté : il manque le protocole permettant aux joueur de se connecter entre eux et certaines fonctionnalités ont été simplifiées à leur état « 2 joueurs » (Par exemple, le score et son affichage ne dépendent pas du nombre de joueurs, de même pour les raquettes) mais la plupart des fonctions d'envoi/réception de paquets sont prévues pour un tableau de « Player ».

La sécurité est convenable, les données sont vérifiées dans les limites du seuil. De plus, même si la phase de connexion des joueurs nécessite un « hôte », la partie ne favorise aucun joueur car elle se fait client à client.

L'interface pourrait subir un lissage de texture car les nôtres ont été conçues de façon archaïque sur paint. Il faudrait également un menu permettant de choisir de créer une partie ou d'en rejoindre une plutôt que d'utiliser la ligne de commande.

Un système de fin de partie (chrono, limite de score) serait également le bienvenu, ainsi qu'un bouton de pause.

Bibliographie

Nous avons écrit tout notre code, internet nous a aidé simplement pour des pistes ou pour les signatures de fonctions.

https://docs.oracle.com/ – Documentation java

http://stackoverflow.com/ – https://openclassroom.com – Tutoriels et réponses aux problèmes

Cours et TDs de java de l'année dernière