

Mini Games

M. FRIEDLI, A. GILLIOZ, J. GUERNE
He-Arc Ingénierie
2000 Neuchatel

14 juillet 2017

Résumé

La HES d'été permet aux étudiants de deuxième année d'étude dans le domaine de l'informatique la possibilité de travailler sur un projet libre dans le but d'approfondir leurs connaissances.

Ce rapport décrit et explique les choix d'implémentations pris dans la réalisation de notre projet : Mini Games.

Une planification des tâches ainsi qu'une spécification du travail ont été réalisés dans le but d'organiser au mieux le temps à disposition.

Table des matières

1	Introduction	3
2	Planification	4
3	Conventions de noms	5
4	LibGDX	6
5	Kryonet	7
6	Architecture logicielle	8
6.1	Diagramme de classe	8
7	Communication réseau	11
7.1	Les paquets	12
7.2	Sécurité	12
7.3	Initialisation de la connexion	12
7.4	Communication en jeu	13
8	Minis jeux	14
8.1	Morpion	14
8.1.1	Description	14
8.1.2	Implémentation	14
8.2	Bataille navale	16
8.2.1	Description	16
8.2.2	Implémentation	16
9	Identité graphique	20
9.1	Interface	20
10	Tests	23
10.1	Client	23
10.1.1	Login	23
10.1.2	Menu des minis jeux	23
10.1.3	Morpion	24

10.1.4	Bataille navale	24
10.2	Serveur	25
10.2.1	Initialisation	25
10.2.2	Morpion	25
10.2.3	Bataille navale	25
11	Problematique	26
12	Améliorations	27
13	Conclusion	28
14	Crédits	29
15	Bibliographie	30

Chapitre 1

Introduction

Mini Games est une application offrant la possibilité de jouer à des minis jeux très classiques tels que le morpion ou la bataille navale en réseau et en multiplateforme. C'est-à-dire que deux personnes (l'une sur son téléphone Android et l'autre sur son ordinateur) auront la possibilité de se défier à une partie de jeu en ligne. Deux grands outils ont été utilisés pour faciliter l'implémentation de ce projet, il s'agit du framework LibGDX, qui a servi à déployer le même programme sur différentes plateformes, et de la bibliothèque KryoNet, qui a, elle, facilité les échanges réseau.

Les jeux listés dans le cahier des charges sont les suivants :

- Morpion
- Bataille navale
- (Bonus : Jeu de dame)

L'objectif principal de ce projet n'est pas de développer des jeux compliqués et très développés mais plutôt de se concentrer sur le multi-plateforme et le réseau.

Chapitre 2

Planification

Nous avons réparti le travail selon les affinités, compétences et disponibilités de chacun.

- Gestion client - serveur : Jonathan Guerne
- Morpions : Jonathan Guerne
- Bataille navale : Anthony Gilloz
- User Interfaces : Jonathan Guerne et Marc Friedli
- Test : Jonathan Guerne, Anthony Gilloz et Marc Friedli
- Rapport : Jonathan Guerne, Anthony Gilloz et Marc Friedli
- Relecture : Marc Friedli

Nous avons ensuite déterminé les principales tâches à réaliser et leur avons donné un ordre de priorité (Figure 2.1).

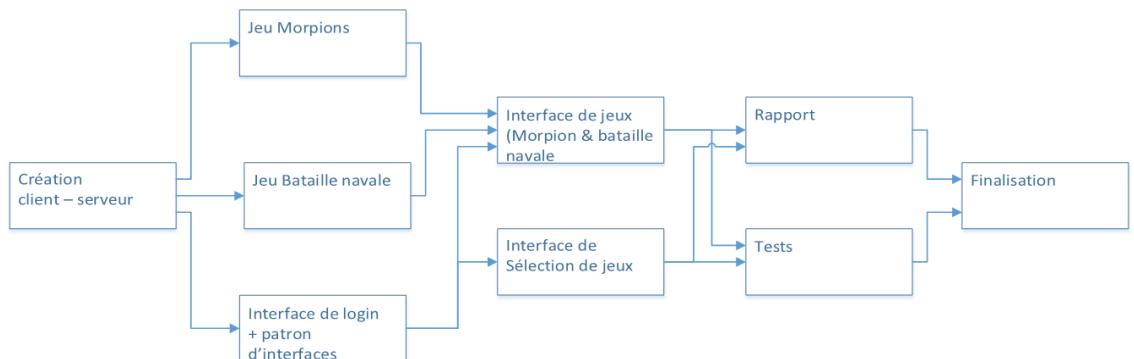


FIGURE 2.1 – Ordre de réalisation des tâches

Chapitre 3

Conventions de noms

Le code a été écrit en respectant la convention camelCase pour les variables et les méthodes. Le camelCase consiste à commencer les noms par des minuscules et si le nom est une composition de plusieurs mots utiliser une majuscule comme séparateur. Les classes respectent la convention PascalCase qui reprend les mêmes conventions que le camelCase excepté le fait que les noms commencent par des majuscules. Les verbes sont privilégiés lors du nommage de méthode par soucis de clarté. L'utilisation de l'anglais plutôt que le français a été préférée.

Chapitre 4

LibGDX



FIGURE 4.1 – Logo de LibGDX

Dès le lancement du projet, nous nous sommes orienté vers libGDX qui est un framework Java gratuit et open source permettant la conception de jeux vidéos. Nous avons fait le choix de travailler avec ce framework en particulier, car nous avions découvert son existence quelque temps auparavant et, en apprenant à le connaître, nous avons découvert à quel point il facilite le déploiement multiplateforme. LibGDX nous a permis de gagner un temps précieux au niveau de l'implémentation puisqu'il propose nativement des fonctionnalités comme la gestion de stages ou de cameras qui aurait, autrement, dû être créés à la main. Étant un framework connu et grandement utilisé, il est simple de trouver des renseignements ou de l'aide concernant sa façon de fonctionner. Nous n'avions aucune expérience dans son utilisation pourtant il ne nous a fallu que très peu de temps avant de commencer à obtenir de bons résultats.

Chapitre 5

Kryonet



FIGURE 5.1 – Logo de KryoNet

Ayant travaillé cette année sur des échanges réseau en Java sans utiliser de bibliothèque externe, nous avons pu constaté que la tâche était fastidieuse. C'est donc naturellement que nous nous sommes tourné vers KryoNet qui est une bibliothèque open source permettant de faciliter la communication entre différents clients. Un des points essentiel du projet était de pouvoir garantir que LibGDX et KryoNet pouvaient cohabiter. Après quelques tests et recherches nous avons pu réaliser que c'était bel et bien le cas (du moins pour le déploiement sur ordinateur et Android).

Chapitre 6

Architecture logicielle

L'architecture logicielle détaille les choix qui ont été pris durant l'implémentation du projet.

Pour optimiser le codage et donc viser à obtenir un code K.I.S.S. (Keep It Simple and Stupid) nous avons choisi de réfléchir sur la logique d'implémentation des classes à l'aide de différents diagrammes avant de commencer le projet. Nous avons donc pu réfléchir aux différents problèmes potentiels et aux différents moyens de rendre de code plus apte à être retravailler plus tard (ajout de fonctionnalités).

6.1 Diagramme de classe

Le diagramme de classe a été réalisé au début du projet dans le but de lui donner une direction. Il a ensuite été généré en fin de projet dans le but de présenter de façon condensé tous les éléments du projet (voir figure 6.1 et 6.2).

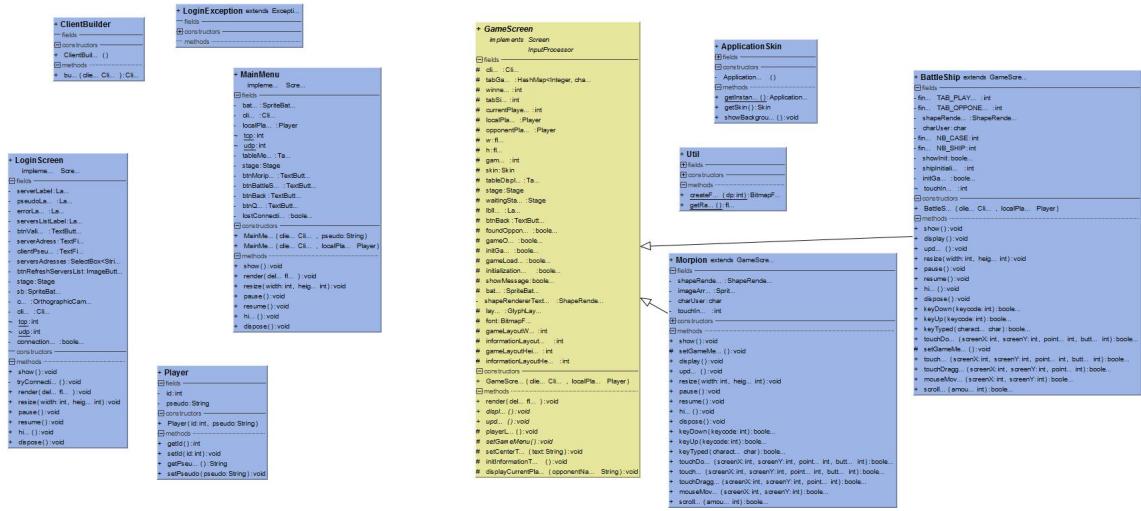


FIGURE 6.1 – Diagramme de classe du programme client

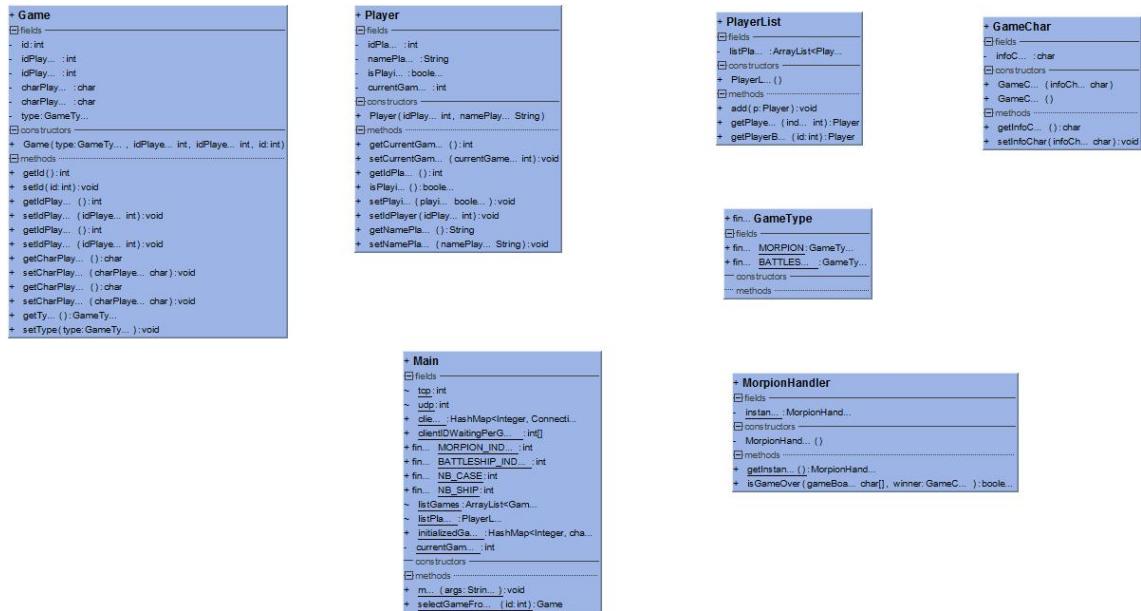


FIGURE 6.2 – Diagramme de classe du programme serveur

L'interface "Screen" fournie par LibGDX permet d'afficher, comme son nom l'indique, des écrans. C'est pourquoi toutes les classes graphiques de ce projet l'implémentent. Le design pattern template method a été utilisé dans l'implémentation des minis jeux, une classe abstraite nommée GameScreen est héritée

par les implémentations concrètes de minis jeux mais un grands nombre de fonctionnalités propre à tous les jeux (message d'attente d'averaire,gestion de l'abandon de l'averaire,...) ont été directement implémentés dans la classe GameScreen.

Chapitre 7

Communication réseau

Comme dit plus haut, le programme est une collection de minis jeux auxquels les clients auront la possibilité de jouer à plusieurs. Il est décomposé en deux parties : la partie client et la partie serveur. La communication entre ces deux parties est facilitée par l'utilisation de KryoNet une bibliothèque Java open source conçu pour gérer les échanges réseau.

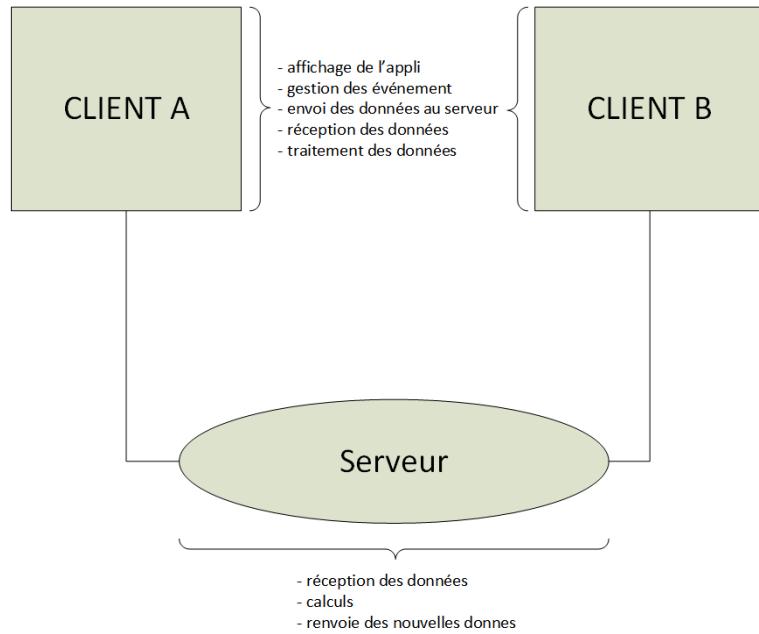


FIGURE 7.1 – Illustration de la communication entre les clients et le serveur

7.1 Les paquets

Les données envoyées entre le client et le serveur transitent sous la forme de paquets, les paquets sont des classes présentent chez le client et le serveur et sont enregistrés au lancement du service.

Pour un client, si une connexion à été établie, il lui est possible d'envoyer par TCP ou UDP des paquets au serveur. Dans le cas complémentaire, pour recevoir les différents paquets émanant du serveur le client met en place un listener qui va automatiquement gérer la réception des paquet et analyser leur contenu. L'implémentation du serveur est identique à la nuance près qu'elle laisse le choix de la personne (de l'adresse) à qui sera envoyé le paquet. En effet toutes les infos, tous les paquets, transitent par le serveur, c'est lui ensuite qui les traitent et les renvoient aux différents clients concernés.

Pour que le serveur et le client puissent communiquer efficacement, il était nécessaire que les paquets utilisent un identifiant bien précis (connu du serveur et du client) pour connaître leur domaine d'utilisation. Les identifiants sont ici de simples integers.

Identifiant du paquet	Domaine d'utilisation
[0-999]	Login et configuration générale
[1000-1999]	paquets liés au morpion
[2000-2999]	paquets liés à la bataille navale

7.2 Sécurité

Pour empêcher aux versions obsolètes de se connecter au serveur, un système de version a été mis en place. Chaque paquet qui transite possède une version, le serveur peut tester si cette version est suffisamment récente avant même d'analyser le contenu du paquet. Si la version n'est pas à jour, le serveur ignorera simplement le paquet. En absence de réponse du serveur le joueur ne pourra donc pas se connecter.

7.3 Initialisation de la connexion

A l'ouverture du programme le client est invité à entrer une adresse de serveur et un pseudo pour tenter ensuite de se connecter. Afin de faciliter l'entrée de l'adresse du serveur la fonction de découverte des hôtes fournie par KryoNet a été utilisée, elle va fournir une liste d'adresse qui seront ensuite présentées à l'utilisateur sous la forme d'une liste déroulante. Si l'utilisateur sélectionne un élément de la liste, l'adresse est automatiquement copié dans le champs de l'adresse du serveur.

Une fois que le serveur a reçu un paquet de login (une tentative de connexion a été envoyée) il stocke le nouveau joueur dans une liste puis retourne au client un paquet de confirmation lui indiquant que la connexion a réussie. Dès la réception du paquet de confirmation, le client peut changer d'écran et afficher le menu de sélection des jeux.

7.4 Communication en jeu

Quand un joueur décide de lancer une nouvelle partie d'un jeu il transmet un paquet au serveur donnant comme information son ID et le jeu auquel il souhaite jouer. Le serveur va, ensuite, vérifier si quelqu'un est déjà en train d'attendre de lancer une même partie. Si c'est le cas, la partie peut commencer ! le serveur prépare donc un paquet de création de partie contenant l'ID et le nom de tous les joueurs et le numéro (également appelé ID) de la partie en elle-même (utile plus tard).

Dans le cas où il n'y a personne dans la liste d'attente, l'ID du client va être stocké dans une variable faisant office de salle d'attente du jeu désiré.. Comme ce projet n'implémente que des jeux à deux joueurs la "salle d'attente" pour jouer à un jeu ne sera jamais composée de plus d'une personne.

Tour à tour les joueurs envoient des informations aux serveurs propre au jeu auquel ils sont en train de jouer qui va ensuite se charger de tester si la partie est terminée ou non et enverra les bons paquets en conséquence.

Quand une partie est finie un message s'affiche chez les joueurs leur indiquant s'ils ont gagné, perdu ou encore fait un match nul (morpion). Ils sont ensuite invité à cliquer sur l'écran pour revenir à la sélection des minis jeux.

Si un joueur quitte la partie alors qu'elle n'est pas terminée le serveur en est informé. Il récupère ensuite les informations de la partie que ce joueur était en train de faire et contact son adversaire pour lui signaler l'abandon de son adversaire.

Chapitre 8

Minis jeux

Dans ce chapitre seront présentés les différents minis jeux mis en place dans ce projet ainsi que leur implémentation. Le serveur garde un historique des jeux sous la forme d'une liste d'objets contenant les informations sur les différents joueurs opposés durant la partie.

8.1 Morpion

8.1.1 Description

Le morpion est un jeu très simple dans lequel deux joueurs s'affrontent sur un plateau de 3 x 3 cases. Chaque joueur possède un caractère (joueur 1 'x' et joueur 2 'o'). Tour à tour, ils vont devoir placer ces caractères dans le plateau de jeu dans le but de faire une ligne horizontale, verticale ou encore diagonale. Si la partie se finit sans qu'aucun joueur n'ai réussi à remplir une ligne/ une diagonale c'est un match nul.

8.1.2 Implémentation

Quand un joueur démarre une partie s'il ne trouve pas d'adversaire il est mis en attente. Un message lui signalant l'attente d'un adversaire s'affiche sur l'écran, une fois le second joueur connecté l'écran de jeu s'affiche (figure 8.1).



FIGURE 8.1 – Attente d'un adversaire

L'écran de jeu est composé en 2 parties : une première, en haut de l'écran contenant des informations sur la partie ainsi qu'un bouton "retour" et une seconde, occupant le reste de l'écran pour le plateau de jeu (figure 8.2).

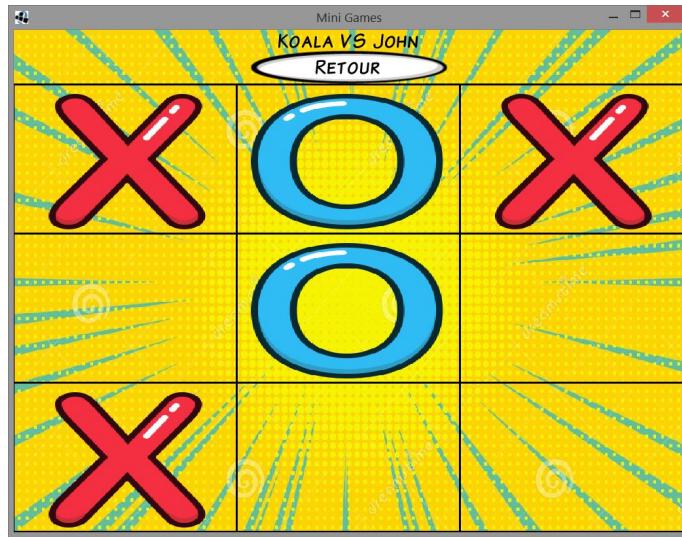


FIGURE 8.2 – Ecran de jeu du morpion

Il s'agit du jeu tour par tour, c'est la raison pour laquelle au lancement du jeu un message est envoyé à chacun des joueurs de la partie leur donnant des in-

dications sur ce qu'ils doivent faire ("c'est votre tour" ou "tour de l'adversaire"). Comme dit plus haut le plateau de jeu est un tableau de 3x3 cases, il est donc nécessaire de savoir dans quelle case le joueur clique avant tout. LibGDX met à disposition des listeners de clics sur l'écran ce qui signifie qu'il sera possible de récupérer les coordonnées X et Y du clic. Une fois ces coordonnées obtenues on les analyse pour savoir quelle est l'index de la case sur laquelle le joueur a cliqué. Il est à noter que la récupération de l'index de la case cliquée ne se fait que chez le joueur dont c'est le tour.

Quand la case est cliquée un paquet contenant les informations du plateau de jeu fraîchement modifié est envoyé au serveur. Celui-ci traitera le plateau de jeu pour savoir si la partie est terminée ou non, en fonction de cette analyse le serveur enverra aux deux joueurs les paquets correspondant (plateau de jeu + changement de joueur ou id du gagnant et plateau de jeu).

8.2 Bataille navale

8.2.1 Description

Notre jeu de bataille navale, reprend les règles normales du jeu mais avec quelques différences sur le placement et la découverte des bateaux (ennemis ou alliés). Au lieu de pouvoir placer des bateaux de taille variable (figure 8.3) comme dans un jeu classique de bataille navale, notre jeu permet de placer un bateau sur une seule et unique case voir figure 8.4. Il s'agira alors pour notre adversaire de trouver nos bateaux sur la grille.

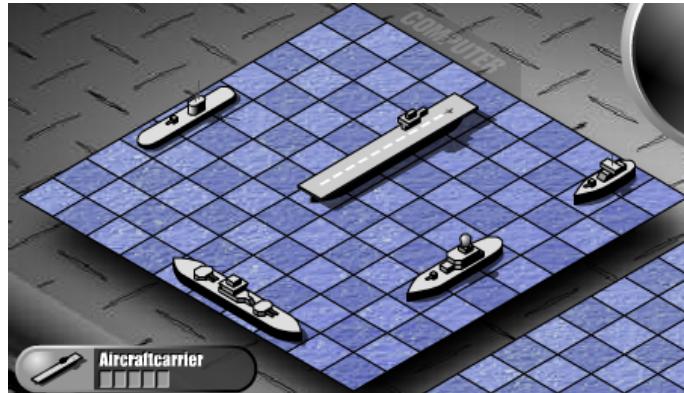


FIGURE 8.3 – Ecran de bataille navale classique

8.2.2 Implémentation

Comme pour le morpion, dès qu'un joueur commence une partie il est mis dans une file d'attente (figure 8.1). Une fois un adversaire trouvé, il y a une phase d'initialisation et à ce moment-là les joueurs vont placer leurs bateaux

(figure 8.4). Une fois l'initialisation confirmée, la partie peut alors commencer et chaque joueur, va à son tour, devoir découvrir les bateaux de son adversaire.



FIGURE 8.4 – Ecran de jeu de la bataille navale

Le joueur à la possibilité d'appuyer où il veut sur le plateau pour trouver les bateaux de son opposant. S'il touche un bateau, celui-ci prend alors feu (figure 8.5). S'il tire à côté, des gouttes d'eau apparaissent (figure 8.6).



FIGURE 8.5 – Bateau en feu



FIGURE 8.6 – Gouttes d'eau

Durant une partie, le joueur a aussi la possibilité de changer le plateau de jeu qu'il veut afficher. Il peut choisir entre celui où il y a ses propres bateaux ou celui avec les endroits où il a tiré.

Une fois que tous les bateaux d'un joueur ont été trouvés (figure 8.7), la partie se termine.



FIGURE 8.7 – Ecran de victoire de la bataille navale

Durant la partie, il y a deux grandes phases de synchronisation. Une première, comme dans le jeu de morpion, et la deuxième au moment où le premier joueur confirme le positionnement de ses bateaux. Il envoie alors un paquet au serveur, celui-ci va stocker l'ID et le tableau des bateaux du joueur. Une fois que les deux joueurs ont confirmés, la partie est lancée et les joueurs, tour à tour vont appuyer pour trouver les bateaux de leur adversaire.

Pour la gestion des bateaux et où le joueur tire, nous utilisons quatre tableaux, chaque joueur en possédant deux. Un premier tableau pour le positionnement des bateaux et un deuxième pour savoir où le joueur a tiré. C'est en parcourant le tableau du positionnement des bateaux et en le testant avec le

tableaux des tirs de l'adversaire que nous pouvons savoir si un bateau à été touché.

Chapitre 9

Identité graphique

Le jeu possède un style fortement inspiré de l'univers des comics books (bandes dessinées principalement américaines), ce choix d'inspiration est principalement lié à l'utilisation d'un très bon thème graphique pour les éléments d'interface de LigGDX. Le thème nous plaisant nous avons choisi d'associer notre jeu à cet univers en adaptant l'interface, les textes et les images. C'est de plus un choix qui sort de l'ordinaire, car il est plutôt rare de voir des jeux reprenant ce style graphique, ce qui est, à notre avis, un point positif pour notre projet.

Concernant l'image utilisée en background, il s'agit d'une image dont on peut acheter le droit de diffusion ou tous les droits directement sur internet. Comme il s'agit d'un projet qui n'a pas pour objectif (actuellement) d'être déployé, nous n'avons pas effectué l'achat. Si nous décidions l'inverse, il faudrait payer.

9.1 Interface

L'interface du programme est découpé en trois parties principales :

- L'écran de login (figure 9.1)
- Le menu de sélection des minis jeux (figure 9.2)
- Le jeu en lui-même (voir figures aux chapitres 8.1 et 8.2)



FIGURE 9.1 – Ecran de login

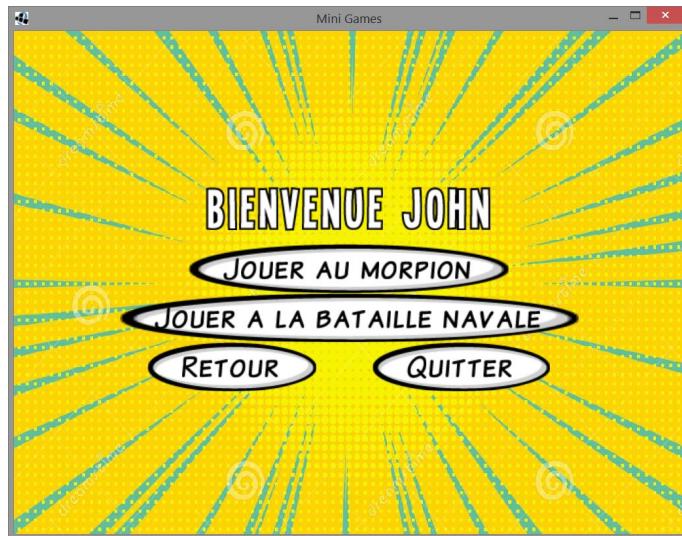


FIGURE 9.2 – Ecran de sélection des jeux

LibGDX fournit des objets "Table", très utile pour organiser des éléments de formulaires sur plusieurs lignes (à la manière d'un tableau HTML) ils ont été utilisés pour permettre de créer l'alignement de l'écran de login et du menu présenté ci-dessus.

Les skins LibGDX sont des collections de ressources graphiques utilisés par

LibGDX pour personnaliser l'aspect des éléments de l'interface. LibGDX fournit un skin par défaut mais comme dit dans l'introduction de ce chapitre nous avons opté pour un skin basé sur le thème des "comics". C'est grâce à lui que nos labels, texts, fields, buttons... ont leur aspect.

Chapitre 10

Tests

Ce chapitre présente les différents tests qui ont été effectués sur le programme.

10.1 Client

10.1.1 Login

- Le programme se lance correctement
- L'utilisateur à le focus sur l'entrée du pseudo
- La touche "Enter" du clavier lance la tentative de connexion
- Le bouton "connexion" lance la tentative de connexion
- Message d'erreur si l'adresse de serveur n'est pas correcte
- Message d'erreur s'il n'y a pas de pseudo
- Message d'erreur si la version n'est pas à jour
- Découverte des serveurs dans le réseau local
- Changement de l'adresse du serveur lors de la sélection d'un des éléments de la liste des serveurs
- Le bouton "refresh" relance la recherche de serveur
- Bonne disposition des éléments graphiques dans la fenêtre

10.1.2 Menu des minis jeux

- Affichage du nom du joueur
- Bonne disposition des éléments graphiques de la fenêtre
- Le bouton "retour" renvoie le client à la page de connexion
- Le bouton "quitter" quitte le programme
- Le bouton "jouer au morpion" lance une partie de morpion
- Le bouton "jouer à la bataille navale" lance une partie de bataille navale

10.1.3 Morpion

- Le joueur est mis en attente s'il est seul à vouloir jouer
- Le texte "Attente d'un autre joueur" s'affiche pendant l'attente d'un adversaire
- Le bouton "retour" renvoie au menu de sélection des jeux
- La partie se lance quand deux joueurs lancent une partie sur le même serveur
- Un texte indique quel joueurs doit jouer
- Si ce n'est pas le tour du joueur, cliquer dans l'air de jeu ne fait rien
- Si c'est le tour du joueur, cliquer dans l'air de jeu informe le serveur du choix de case
- Les joueurs reçoivent le plateau de jeu mis à jour par le serveur
- Si la partie est terminée, les deux joueurs reçoivent des infos du serveur signifiant la fin de la partie
- Le texte "vous avez gagné" s'affiche chez le gagnant
- Le texte "vous avez perdu" s'affiche chez le perdant
- Le texte "égalité" s'affiche en cas d'égalité
- Quand la partie est terminé si les joueurs cliquent sur l'écran ils retournent au menu de sélection des minis jeux
- Le texte "votre adversaire a quitté la partie" s'affiche si l'adversaire du joueur a quitté la partie

10.1.4 Bataille navale

- Le joueur est mis en attente s'il est seul à vouloir jouer
- Le texte "Attente d'un autre joueur" s'affiche pendant l'attente d'un adversaire
- Le bouton "retour" renvoie au menu de sélection des jeux
- La partie se lance quand deux joueurs lancent une partie sur le même serveur
- Le texte avertit que les joueurs sont en phase d'initialisation (ils peuvent poser leur bateaux)
- Au clic du bouton "confirmer", un message apparaît si tous les bateaux ne sont pas posé, sinon la partie commence
- Un texte indique quel joueur doit jouer
- Le plateau de jeu se met à jour depuis le serveur
- Si un joueur touche un bateau ennemi alors celui-ci devient en feu
- Si un joueur gagne la partie un message apparaît pour avertir le gagnant et le perdant
- Quand la partie est terminée si les joueurs cliquent sur l'écran ils retournent au menu de sélection des minis jeux
- Le texte "votre adversaire a quitté la partie" s'affiche si l'adversaire du joueur a quitté la partie

10.2 Serveur

10.2.1 Initialisation

- Le serveur se lance correctement
- Des clients peuvent se connecter au serveur
- le serveur peut recevoir des Packets des clients
- le serveur peut envoyer des paquets aux clients
- Les joueurs sont sauvegardés dans une liste
- Les games (les jeux) sont sauvegardés dans une liste

10.2.2 Morpion

- Si personne n'est en attente sur ce jeu le joueur est mis en attente
- Si un joueur est déjà en attente une partie est lancée
- Si un joueur quitte la partie quand il est en attente il n'est plus en attente
- Le serveur reçoit des paquet donnant des informations sur le plateau de jeu de la part des joueurs
- Le serveur test si la partie est finie ou non
- Le serveur renvoie de nouvelle infos sur la partie aux deux joueurs
- Si un joueur se déconnecte ou s'il appuie sur le bouton "retour" le serveur est notifié
- Le serveur envoie un paquet au joueur dont l'adversaire a quitté la partie

10.2.3 Bataille navale

La partie de la bataille navale se comporte comme la partie du morpion, sauf pour la deuxième phase d'initialisation.

- Si un joueur n'a pas encore validé la position de ses bateaux, alors l'autre joueur est mis en attente
- Si un joueur à déjà validé la position de ses bateaux alors la partie commence

Chapitre 11

Problematique

Une des problématique rencontrée a été la mise en place d'un serveur sur Android. En effet, LibGDX "facile" le portage de l'application client vers les périphériques Android mais le serveur a lui été développé sans libGDX. Une ébauche d'application Android opérant de la même manière que le serveur Java classique a été mise en place. L'implémentation s'est avérée très lourde et peu fructueuse dans le sens où lorsqu'un client se connectait à un serveur Android celui-ci crashait immédiatement.

Dans de très rares cas il est également possible que le lancement de la partie ne s'effectue pas tout à fait correctement. Un des deux joueurs aura déjà l'écran du début de partie alors que l'autre affiche toujours l'écran d'attente. Cela n'est cependant pas un problème majeur car si ce problème survient les deux joueurs ont accès à un bouton "retour" leur permettant de revenir à la sélection des minis jeux et de relancer une partie.

Un autre problème que nous avons rencontré est le fait que, pour des raisons de performances, LibGDX n'est pas thread-safe, ce qui fait que, en fonction de l'ordre d'arrivée des paquets, il est possible que le programme tente d'accéder à une ressource qui n'existe pas encore. Il s'agit d'un problème connu de LibGDX qui, pour l'instant, refuse de le corriger afin de garder leurs performances. Cependant, ce problème n'arrivant que lors de très rares occasions, il aurait été trop couteux en terme de ressources de le corriger et s'assurer que le fix était bien effectif, car nous n'avons pas moyen de trouver quel appel devrait être exécuté sans changement de contexte (atomique).

Le programme requiert également une grande quantité de mémoire ce qui nous laisse penser que tout n'est pas supprimé comme il devrait l'être.

Chapitre 12

Améliorations

La communication entre pc et Android est fonctionnelle mais, comme expliqué plus haut, notre shouhait était d'étendre notre proramme à encore deux autres plate-formes : IOS et HTML5 (navigateur web). Certains problèmes d'implémentations pourraient survenir (utilisation d'outils non supportés sur certaines plate-formes, etc.) mais avec du temps, l'application pourrait être totalement multi-plateforme. De nouveaux jeux pourraient aussi être ajoutés tels que, comme décrit dans notre cahier des charges, un jeu de dame. Un système de points récompensant le joueur vainqueur. Les points pourraient être utilisés pour mettre en place un classement des meilleurs joueurs du serveur. Une musique pourrait être ajoutée avec un bouton de settings permettant de changer de pseudo.

Chapitre 13

Conclusion

Nous avons pu réaliser les objectifs primaires que nous nous étions fixés dans le cahier des charges, nous sommes donc satisfait du résultat. Une de nos plus grosse attente était de pouvoir continuer d'apprendre à utiliser des outils tel que LibGDX ou KryoNet et nous avons été agréablement surpris de la facilité de mise en place de ces deux outils dans un même environnement.

Chapitre 14

Crédits

- Image de bateau pirate
<http://www.flaticon.com>
- Image de background
<https://www.dreamstime.com>
- Image de flamme
<http://www.pixabay.com>
- Image de goutte d'eau
<http://www.clipground.com>
- Image de X (morpion)
<http://www.iconfinder.com>
- Image de O (morpion)
<http://www.iconfinder.com>
- Skin LibGDX
<https://github.com/czyzby.gdx-skins/tree/master/comic>

Tous les sites cités ci-dessus ont été consultés durant la semaine du 10.07.2017

Chapitre 15

Bibliographie

- <https://github.com/libgdx/libgdx/issues/3491>
- <http://libgdx.badlogicgames.com/nightlies/docs/api/>
- <https://github.com/EsotericSoftware/kryonet>
- Cours de Java de Monsieur Carrino