PROJET JAVA 2TL2 : RAPPORT

BATAILLE NAVALE

Groupe n° 13 Conotte Sébastien, Gilles Timothy, Lefevre Alexandre

Lien gitHub : : https://github.com/Projet2TL2/Projet

Table des matières

Cahier des charges

Objectif :

Pour appliquer les connaissances acquises pendant le cours de programmation en Java, nous avons comme objectif de programmer un jeu. Il s'agit du jeu « Bataille navale » alias « toucher-couler ».

Nous allons aussi créer un compte GitHub sur lequel nous déposerons notre code source ainsi qu'une page wiki sur ce même site.

Descriptif Client :

Nous désirons mettre en place une application qui permettra de jouer à bataille navale contre l'ordinateur ou contre un second joueur. Il y aura donc un menu permettant de choisir quel mode de jeux choisir et dans lequel on pourra gérer ces options.

Fonctionnalité :

Il est possible de jouer contre l'ordinateur en console ou contre un autre joueur sur une interface. Une partie dure approximativement 5 minutes. Il y a 3 possibilité : soit touché, soit coulé ou encore raté. Chaque option a sa propre couleur afin de les distinguer .

Plateforme de développement :

La documentation du projet est disponible sur le site internet : https://github.com/Projet2TL2/Projet
La documentation est composée de plusieurs sections :

- Accueil : accueil du site, avec diverses informations sur le projet.

-Wiki : section wiki, permettant de suivre l'état d'avancement du projet ainsi que d'autres ressources.

-Source : la section source permet du consulter les sources du projet.

Quelques fonctionnalités attendues avant l'implémentation du projet:

-Possibilité de choisir sa faction/nation. Chacune d'entre elle aura un attribut. Type de bateau différent, attaques différentes.

-Défi à réaliser : par exemple si vous toucher 3 fois de suite, vous aurez un tour bonus.

-Effets sonores et visuel : quand on touche un bruit d'explosion et une petite flamme sur la case, couler…

- Attaque spéciale : par exemple si un joueur a beaucoup de retard dans la partie il a droit à 2 tirs, faire une sorte toutes les 5 fois ou on est touché, on a droit à un tir plus puissant.

Lien GitHub :

https://github.com/Projet2TL2/Projet

Evolution du cahier des charges

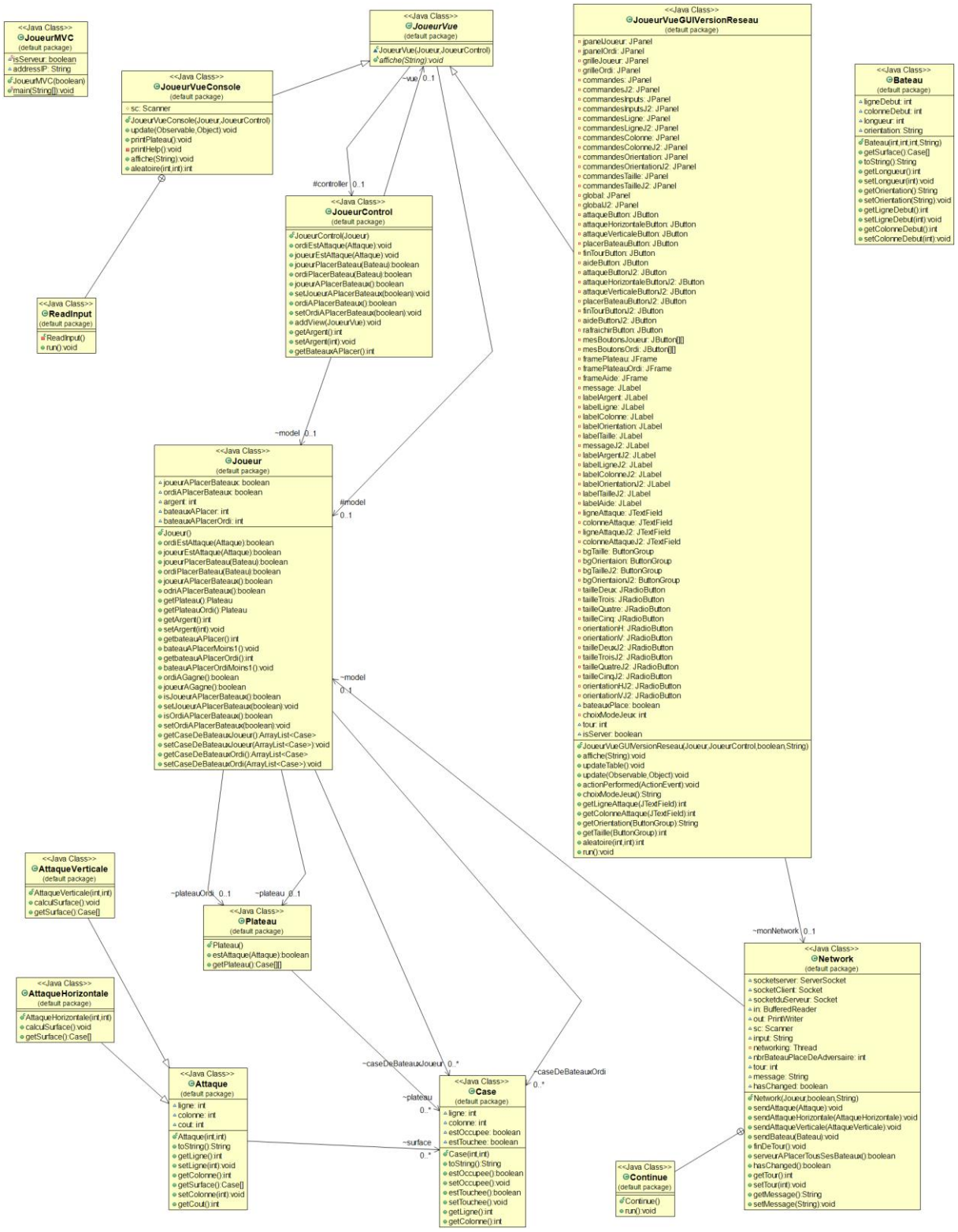
Au vu de la complexité du projet et du manque de temps, nous avons dus effectuer quelques modifications sur le contenu de notre application. Bien que dans l'ensemble le rendu de l'application soit conforme aux attentes.

Nous avons retiré la notion de thème (de choix d'équipe) ainsi que les différentes spécificités de ces 2 classes pour nous concentrer sur le bon fonctionnement de notre code.

D'autres éléments tels que les bonus supplémentaires et les effets (sonore ou autre) ont également été mis de côté.

Diagramme UML

Le diagramme UML à beaucoup évoluer au cours du projet. En effet, nous avons débuter notre projet avec nos connaissances de java et nous avons du changer la disposition de nos classes et l'implémentation de notre code pour passer au model MVC vu seulement au tp7. Pour finit, nous avons du compléter notre code MVC avec une gestion de jeux en réseau vu au tp9.


<<Java Class>>
JoueurMVC
(default package)
isServeur: boolean
addressIP: String
JoueurMVC(boolean)
main(String[]):void
<<Java Class>>
JoueurVue
(default package)
JoueurVue(Joueur,JoueurControl)
affiche(String):void
~vue 0..1
<<Java Class>>
JoueurVueGUIVersionReseau
(default package)
jpaneJoueur: JPanel
jpaneOrdi: JPanel
grilleJoueur: JPanel
grilleOrdi: JPanel
commandes: JPanel
commandesJ2: JPanel
commandesInputs: JPanel
commandesInputsJ2: JPanel
commandesLigne: JPanel
commandesLigneJ2: JPanel
commandesColonne: JPanel
commandesColonneJ2: JPanel
commandesOrientation: JPanel
commandesOrientationJ2: JPanel
commandesTaille: JPanel
commandesTailleJ2: JPanel
global: JPanel
globalJ2: JPanel
attaqueButton: JButton
attaqueHorizontaleButton: JButton
attaqueVerticaleButton: JButton
placerBateauButton: JButton
finTourButton: JButton
aideButton: JButton
attaqueButtonJ2: JButton
attaqueHorizontaleButtonJ2: JButton
attaqueVerticaleButtonJ2: JButton
placerBateauButtonJ2: JButton
finTourButtonJ2: JButton
aideButtonJ2: JButton
rafraichirButton: JButton
mesBoutonsJoueur: JButton[][]
mesBoutonsOrdi: JButton[][]
framePlateau: JFrame
framePlateauOrdi: JFrame
frameAide: JFrame
message: JLabel
labelArgent: JLabel
labelLigne: JLabel
labelColonne: JLabel
labelOrientation: JLabel
labelTaille: JLabel
messageJ2: JLabel
labelArgentJ2: JLabel
labelLigneJ2: JLabel
labelColonneJ2: JLabel
labelOrientationJ2: JLabel
labelTailleJ2: JLabel
labelAide: JLabel
ligneAttaque: JTextField
colonneAttaque: JTextField
ligneAttaqueJ2: JTextField
colonneAttaqueJ2: JTextField
bgTaille: ButtonGroup
bgOrientaion: ButtonGroup
bgTailleJ2: ButtonGroup
bgOrientaionJ2: ButtonGroup
tailleDeux: JRadioButton
tailleTrois: JRadioButton
tailleQuatre: JRadioButton
tailleCinq: JRadioButton
orientationH: JRadioButton
orientationV: JRadioButton
tailleDeuxJ2: JRadioButton
tailleTroisJ2: JRadioButton
tailleQuatreJ2: JRadioButton
tailleCinqJ2: JRadioButton
orientationHJ2: JRadioButton
orientationVJ2: JRadioButton
bateauxPlace: boolean
choixModeJeux: int
tour: int
isServer: boolean
JoueurVueGUIVersionReseau(Joueur,JoueurControl,boolean,String)
affiche(String):void
updateTable():void
update(Observable,Object):void
actionPerformed(ActionEvent):void
choixModeJeux():String
getLigneAttaque(JTextField):int
getColonneAttaque(JTextField):int
getOrientation(ButtonGroup):String
getTaille(ButtonGroup):int
aleatoire(int,int):int
run():void
<<Java Class>>
Bateau
(default package)
ligneDebut: int
colonneDebut: int
longueur: int
orientation: String
Bateau(int,int,int,String)
getSurface():Case[]
toString():String
getLongueur():int
setLongueur(int):void
getOrientation():String
setOrientation(String):void
getLigneDebut():int
setLigneDebut(int):void
getColonneDebut():int
setColonneDebut(int):void
<<Java Class>>
JoueurVueConsole
(default package)
sc: Scanner
JoueurVueConsole(Joueur,JoueurControl)
update(Observable,Object):void
printPlateau():void
printHelp():void
affiche(String):void
aleatoire(int,int):int
#controller 0..1
<<Java Class>>
JoueurControl
(default package)
JoueurControl(Joueur)
ordiEstAttaque(Attaque):void
joueurEstAttaque(Attaque):void
joueurPlacerBateau(Bateau):boolean
ordiPlacerBateau(Bateau):boolean
joueurAPlacerBateaux():boolean
setJoueurAPlacerBateaux(boolean):void
ordiAPlacerBateaux():boolean
setOrdiAPlacerBateaux(boolean):void
addView(JoueurVue):void
getArgent():int
setArgent(int):void
getBateauxAPlacer():int
<<Java Class>>
ReadInput
(default package)
ReadInput()
run():void
~model 0..1
<<Java Class>>
Joueur
(default package)
joueurAPlacerBateaux: boolean
ordiAPlacerBateaux: boolean
argent: int
bateauxAPlacer: int
bateauxAPlacerOrdi: int
Joueur()
ordiEstAttaque(Attaque):boolean
joueurEstAttaque(Attaque):boolean
joueurPlacerBateau(Bateau):boolean
ordiPlacerBateau(Bateau):boolean
joueurAPlacerBateaux():boolean
odriAPlacerBateaux():boolean
getPlateau():Plateau
getPlateauOrdi():Plateau
getArgent():int
setArgent(int):void
getbateauAPlacer():int
bateauAPlacerMoins1():void
getbateauAPlacerOrdi():int
bateauAPlacerOrdiMoins1():void
ordiAGagne():boolean
joueurAGagne():boolean
isJoueurAPlacerBateaux():boolean
setJoueurAPlacerBateaux(boolean):void
isOrdiAPlacerBateaux():boolean
setOrdiAPlacerBateaux(boolean):void
getCaseDeBateauxJoueur():ArrayList<Case>
setCaseDeBateauxJoueur(ArrayList<Case>):void
getCaseDeBateauxOrdi():ArrayList<Case>
setCaseDeBateauxOrdi(ArrayList<Case>):void
#model 0..1
~model 0..1
<<Java Class>>
AttaqueVerticale
(default package)
AttaqueVerticale(int,int)
calculSurface():void
getSurface():Case[]
<<Java Class>>
AttaqueHorizontale
(default package)
AttaqueHorizontale(int,int)
calculSurface():void
getSurface():Case[]
~plateauOrdi 0..1
~plateau 0..1
<<Java Class>>
Plateau
(default package)
Plateau()
estAttaque(Attaque):boolean
getPlateau():Case[][]
~monNetwork 0..1
<<Java Class>>
Network
(default package)
socketserver: ServerSocket
socketClient: Socket
socketduServeur: Socket
in: BufferedReader
out: PrintWriter
sc: Scanner
input: String
networking: Thread
nbrBateauPlaceDeAdversaire: int
tour: int
message: String
hasChanged: boolean
Network(Joueur,boolean,String)
sendAttaque(Attaque):void
sendAttaqueHorizontale(AttaqueHorizontale):void
sendAttaqueVerticale(AttaqueVerticale):void
sendBateau(Bateau):void
finDeTour():void
serveurAPlacerTousSesBateaux():boolean
hasChanged():boolean
getTour():int
setTour(int):void
getMessage():String
setMessage(String):void
~caseDeBateauxJoueur 0..*
~caseDeBateauxOrdi 0..*
<<Java Class>>
Attaque
(default package)
ligne: int
colonne: int
cout: int
Attaque(int,int)
toString():String
getLigne():int
setLigne(int):void
getColonne():int
getSurface():Case[]
setColonne(int):void
getCout():int
~plateau 0..*
~surface 0..*
<<Java Class>>
Case
(default package)
ligne: int
colonne: int
estOccupee: boolean
estTouchee: boolean
Case(int,int)
toString():String
estOccupee():boolean
setOccupee():void
estTouchee():boolean
setTouchee():void
getLigne():int
getColonne():int
<<Java Class>>
Continue
(default package)
Continue()
run():void

Les choix d'implémentations effectués

Nous avons utilisé des arraylists dans la classe joueur.

2 arraylist de case. Le premier pour les cases bateau du joueur et l'autre pour les cases bateau de l'ordinateur. Les cases correspondent à la surface occupée par un bateau.

Pour quelle raison ? Car il est plus simple de rechercher une case dans une arrayList que dans un tableau.

Pour l'interface graphique, nous avons décidé de représenter le plateau de jeu par un tableau de 11 sur 11 formé de JButtons avec 1 colonne et 1 ligne représentants les indices des JButtons.

L'utilisation de JButtons nous permet de directement récupérer les coordonnées de l'endroit où l'utilisateur à cliquer.

A chaque lancement de partie, un thread est créé pour le serveur et pour le client. Les threads sont créés dans la classe Network.

Difficultés rencontrées

Lors de notre avancée dans le projet, nous nous sommes vite rendu compte que la quantité de travail nécessaire à la réalisation de notre cahier de charge était trop importante par rapport au temps dont nous disposions.

De plus, toutes les connaissances nécessaires à la réalisation du projet n'étaient pas encore à notre portée lors des premières semaines. Notre travail à évoluer en fonction de la matière aborder lors des séances de travaux pratiques.

Par exemple les « sockets », un des éléments à intégrer à notre projet étaient particulièrement compliquer à implémenter dans notre code vu la semaine tardive à laquelle nous avons découvert la matière.

Notre projet devait pouvoir fonctionner sous le modèle MVC. Or cette partie de la matière n'as été vu que à partir du tp7. Etant donné que le modèle MVC demande une modification importante de notre code de départ, nous avons prit quelques semaines a faire une bonne transition fonctionnelle.

L'interface graphique est peut-être la partie qui nous a pris le plus de temps à mettre en place dans ce projet. Il nous fallait comprendre le concept des jFrame et jPanel pour pouvoir les placer aux bons endroits.Comprendre le placement des composants dans une jframe. Le fonctionnement de boxLayout, etc.

Un problème majeur rencontré dés les premiers jours de notre modèle MVC est que nos actions ne s'effectuaient que 1 action après l'action voulue. Nous avons trouver notre erreur dans le mauvais placement des « NotifyObservers() » dans le model Joueur.

On a compris que le setChanged() et le notifiedObserver() était mal placé et devait se faire après que l'action a eu lieu et avant le return true .

Pistes d'amélioration éventuelles

Au niveau du contenu :

De nombreuses améliorations peuvent être ajoutées à notre programme pour le rendre encore plus attractif.

Fonctionnalité supplémentaire :

1) Le joueur aura la possibilité de choisir une faction. Chaque faction possèdera ses propre atout et faiblesse.
2) Ajuster la taille du plateau de jeu pour permettre des parties plus longues
3) Ajout d'une « intelligence artificiel » pour l'ordinateur, histoire de rendre les parties plus compliquées.

Bateau supplémentaire :

1) En fonction de la faction choisie, le joueur disposera d'un quartier général de forme différente.
2) Différents navires peuvent se rajouter avec des fonctionnalités spéciales. (Ex : un sous-marin qui se cache sous l'eau un tour sur deux …)

Armes supplémentaires :

1) Ajout d'arme spécial en fonction de la faction rejoint par le joueur. Cette arme aura un coup supérieur aux autres attaques.
2) Un radar qui permet de débusquer les bateaux ennemis dans un certain périmètre.

Au niveau de l'interface :

Nous avons essayé de rendre notre interface graphique le plus clair et pratique possible, mais il est toujours possible de l'améliorer. Que cela soit en ajoutant des images ou autres textures qui pourront rendre notre jeu plus réaliste ou encore d'éventuel animation ou effet sonore.

Conclusions individuelles :

Par Sébastien C.

Projet très intéressant à réaliser ! Le fait de devoir travailler en équipe est une bonne chose pour nous habituer à notre futur métier. J'ai appris de très intéressant nouveau concept en réalisant notre jeux comme les Sockets et le placement détaillé dans une JFrame. Je suis un peu déçu de ne pas avoir su réaliser toutes les fonctionnalités du cahier de charge initial mais très satisfait du rendu final avec nos différents soucis rencontrés lors de la réalisation de ce projet.

Par Timothy G.

Ce projet m'as permis de m'améliorer en JAVA et de comprendre des concepts essentiels à la programmation en JAVA. Nous avons appris à nous servir d'outil qui nous aiderons dans notre vie professionnelle future tels que GIT, GitHub. Le projet nous pousse également à comprendre et apprendre une partie de la matière par nos propres moyens, c'est très bien de nous pousser à utiliser nos propres ressources pour améliorer notre niveau en programmation. J'ai également acquis de l'expérience dans la réalisation de travail par groupe. Dans l'ensemble je suis très satisfait de notre application finale.

Par Alexandre L.

Le projet était fort compliqué, à cause du temps que nous avions pour le réaliser et le fait que nous avions d'autres projets et interro à faire. Mais cela nous a appris à travailler par nous-même et en équipe, même si cela n'a pas été une tâche facile. Nous avons eu des problèmes pour donner suite au MVC qui nous obligé à faire marche arrière, malgré cela je trouve que le travail rendu est à la hauteur de nos espoirs, même si tout ce que nous avions imaginé n'a pas été possible à coder.