Rapport

Projet JAVA 2

Dice Wars

Groupe E4

Tom CHANG

Guillaume THIBAUT

Abdel BEKHOUCHE

2020-2021

Sommaire

I. Introduction

Nous avons le plaisir de vous présenter ce projet réalisé en Java. Rappelons tout de même son objectif. Ce projet nous demande la réalisation d'un jeu nommé « **Dice Wars** ». Le but du jeu est simple : plusieurs joueurs possèdent des territoires avec des dés. Ces territoires sont répartis aléatoirement sur une carte. L'objectif de chaque joueur est de conquérir tous les territoires de la carte. Pour cela, le jeu se joue par tour et à chaque tour, un joueur peut sélectionner un de ses territoires pour attaquer ses voisins ennemis. Comme dit précédemment, chaque territoire possède des dés. Ces dés vont nous permettre de créer une bataille qui se règle par le joueur attaquant dont la somme de ses dés jetés est supérieure ou non à celle de l'ennemi. Le résultat décidera de s'il capture ou non le territoire. Ainsi va la logique du jeu jusqu'à qu'il n'y ait plus qu'un seul joueur sur la carte. C'est donc la **Version 1**. Le jeu se joue sur console, peu pratique.

Le projet ne s'arrête pas là, on nous demande aussi d'implémenter une interface graphique (qu'on appellera **Version 2**) permettant de jouer à ce jeu plus facilement avec la souris. Le choix de la bibliothèque d'interface est libre, nous verrons plus tard laquelle nous avons choisi.

II. Version 1

A) Conception

Pour réaliser ce jeu, nous devons séparer en plusieurs étapes. Ces étapes suivent en théorie le cahier des charges du sujet :

La classe Joueur :

Elle doit contenir des informations relatives à chaque joueur comme ses territoires conquis ainsi qu'un identifiant unique. Non seulement cela, il doit pouvoir attaquer un territoire à partir d'un de ses territoires à condition que le territoire en question possède plus d'un dé (Force 1). Le joueur doit aussi pouvoir passer son tour.

La classe Territoire :

Chaque territoire possède un identifiant unique. Pour chaque territoire, il va falloir lui attribuer un joueur, une force (les dés), ainsi que ses voisins pour savoir quels territoires sont connectés et donc attaquable par exemple.

La classe Carte :

C'est l'une des parties majeures du jeu. En effet, elle gère la carte des territoires, se charge de distribuer aléatoirement les territoires aux joueurs, ainsi que les dés à ces territoires, tout ça de manière très aléatoire. La carte dépendra inévitablement de la prochaine classe majeure : Partie.

La classe Partie :

Le jeu tient sur cette classe où l'on organise pratiquement tous les évènements dans une boucle : initialisation de la carte, génération des joueurs et ordre des tours aléatoires, sélection des actions comme attaquer un territoire ou passer son tour et enfin, pour pouvoir visualiser le jeu, afficher la carte à chaque fin d'actions. La boucle se termine lorsqu'un joueur gagnant est déterminé, là où dans quel cas une demande pour relancer une nouvelle partie s'affichera.

B) Réalisation

La classe Territoire s'explique de lui-même dans sa réalisation à partir du concept, attardons-nous plutôt vers les étapes plus complexes. Tout d'abord, commençons par l'initialisation aléatoire de la carte. Il a été nécessaire de décomposer en plusieurs étapes afin d'aboutir à une carte aléatoire :

- Première étape, la création de la matrice est nécessaire. Ce n'est cependant pas n'importe quel matrice. L'objectif était de créer une carte contenant des territoires dispersés formant un pattern vraiment aléatoire. Or remplir complètement une matrice la rendrait rectangulaire. Pour réaliser cela, le choix du parcours du labyrinthe a été utilisé : à partir d'une matrice de base, ici 9x9, il faudra partir d'un point départ aléatoire et avancer partout aléatoirement. La raison est qu'on ne veut en aucun cas un territoire isolé, ce qui causerait la partie à avoir la possibilité de ne jamais se finir. **Le parcours du labyrinthe assure que tous les territoires soient connectés**. Toutes les cases ne sont pas remplies, 30 cases devraient suffire pour le nombre maximal de joueur que nous imposons à notre programme, qui est de 4 joueurs.

- Deuxième étape, attribuer aléatoirement les territoires aux joueurs. Pour cela, nous devons respecter la règle qui dit que tous les joueurs doivent avoir le même nombre total de dés. Le nombre de territoires peut être différent cependant. Pour cela, il a fallu générer une liste aléatoire contenant 30 entiers représentant les ID des joueurs, et ces 30 entiers seront répartis sur les 30 territoires aléatoirement. Cette approche nécessite de calculer le nombre minimum de territoires qu'un joueur doit posséder : nombre de dés divisé par 8, la force maximale d'un territoire. De cette façon, aucun joueur n'aura pas assez de territoires, en tout cas dans notre limite de 4 joueurs. Nous donnons de base 1 dé à chaque territoire.

- Troisième étape, imaginons que chaque joueur possède un pot contenant le restant des dés à disperser. Chaque joueur éparpillera les dés de son pot sur ses territoires.

A partir d'ici, la carte est finalisée et prête à l'emploi pour la Partie :

```
-1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    -1[0](0)    26[2](2)    28[2](4)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    25[3](3)    22[2](1)    27[3](6)    29[3](2)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    11[2](2)    10[3](3)    9[2](2)     8[1](2)     23[1](6)    24[2](1)    -1[0](0)    -1[0](0)
-1[0](0)    12[1](4)    21[3](4)    -1[0](0)    6[3](3)     7[3](3)     -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    13[1](1)    20[1](4)    19[1](2)    5[3](3)     -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    14[2](2)    15[1](3)    18[2](4)    4[3](3)     1[2](6)     -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    -1[0](0)    16[1](1)    17[2](5)    3[1](6)     0[2](1)     -1[0](0)    -1[0](0)    -1[0](0)
-1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    -1[0](0)    2[1](1)     -1[0](0)    -1[0](0)    -1[0](0)
```

Figure 1: La carte

Suivant le concept, le joueur aura la capacité à choisir son action dans la boucle de Partie. Une des actions est attaquée : il doit d'abord choisir un de ses territoires capables d'attaquer et choisir un territoire ennemi cible. Pour réaliser cela, nous devons récupérer ces deux

territoires dans Joueur. Une fois récupérée, il suffira de prendre leur force, générer deux listes contenant des entiers allant de 1 à 6 le nombre de fois que la force du territoire. Ainsi, en faisant le total de tous les entiers pour simuler le lancer de dés, nous pouvons facilement savoir qui a gagné. Au cas où, l'attaquant gagne, il va falloir ajouter l'ID du territoire ennemi dans ses territoires, l'enlever des territoires ennemies, et mettre sa force égale à celui qui l'a attaqué moins 1. Pour rappel, un territoire à 1 force ne peut attaquer, donc ne peut jamais tomber à 0.
Par mesure de sécurité, l'utilisateur ne peut sélectionner un territoire qui n'est pas à lui et ne peut sélectionner l'attaque vers un territoire à lui puisqu'on peut établir une simple liste contenant les territoires capables d'attaquer, et les territoires éligibles pour être attaqués.

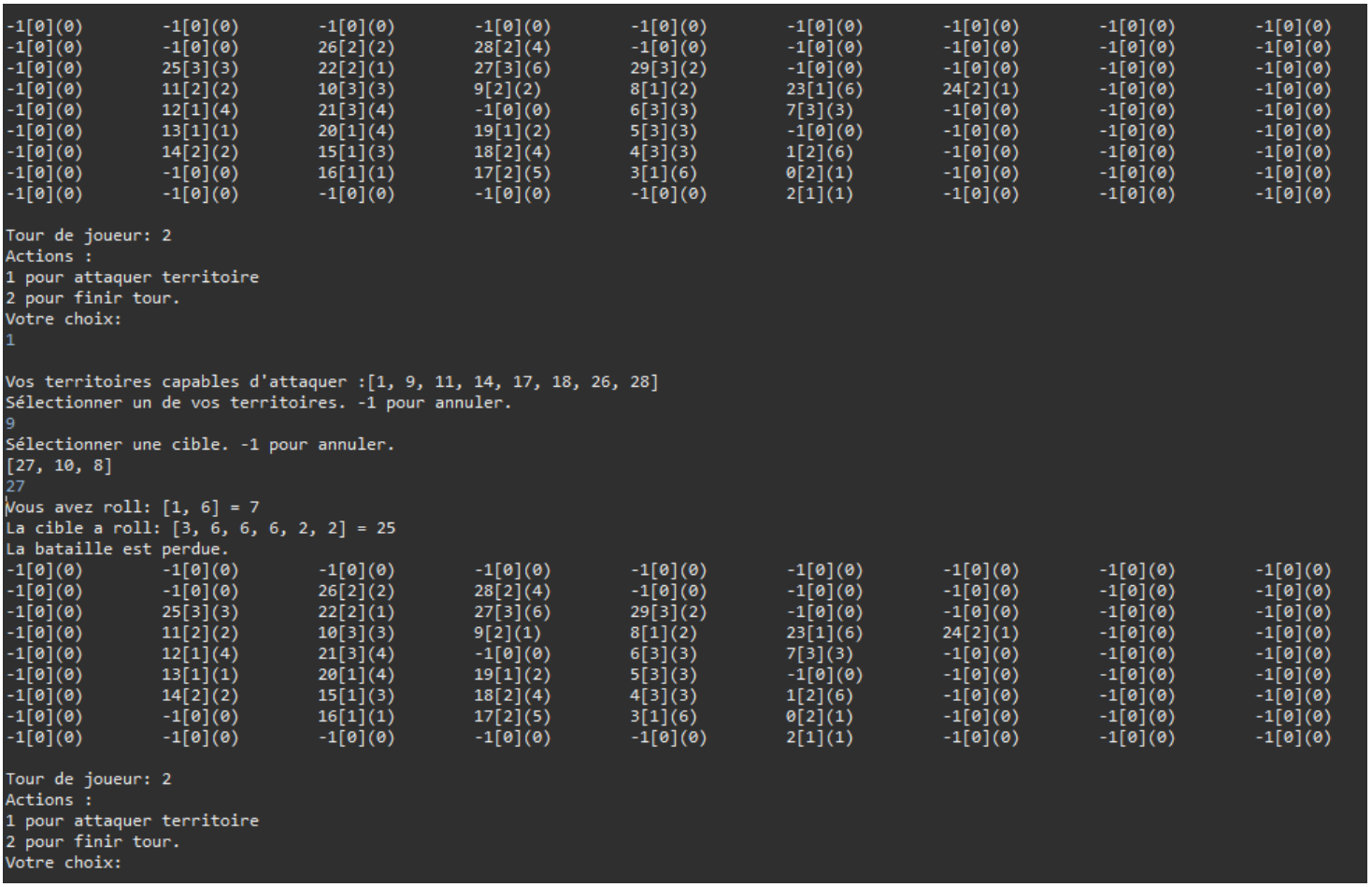

Figure 2: Une partie en cours

La fin du jeu reprend le concept. Pour chaque joueur possédant aucun territoire sera retiré des tours, jusqu'à que la liste des tours devient de taille 1, signifiant qu'il ne reste plus qu'un joueur. On signal alors la victoire de ce joueur.

Voici donc la **Version 1** de Dice Wars.

III. Version 2

Figure 3: Logo personnalisé

Imprévisible. A la vue de ce projet, la manière de jouer à ce jeu nous a rappelés grandement ces grands jeux contenant des cases comme Fire Emblem, ou notre sélection : Azur Lane. Nous avons donc réalisé un jeu basé sur cet univers. Pour présenter très rapidement cet univers, c'est un monde où les navires de la seconde guerre mondiale sont humanisés de la manière la plus stéréotypée du Japon par des personnages de style manga.

Figure 4: Menu principal

Concernant la bibliothèque d'interface graphique, nous avons préféré sélectionner JavaFX plutôt que Swing, pour deux raisons : la familiarité avec **JavaFX** car déjà utilisé dans le passé, et ce dernier est plus récent que Swing ou AWT. Cela nous a permis de créer une interface plus facilement notamment grâce à **Scene Builder** de Gluon. L'aisance d'utilisation des fichiers FXML et le CSS, ainsi que la gestion des scènes par la même occasion lui donnent notre choix.

L'utilisateur aura tout d'abord le choix de jouer directement ou de paramétrer sa partie :

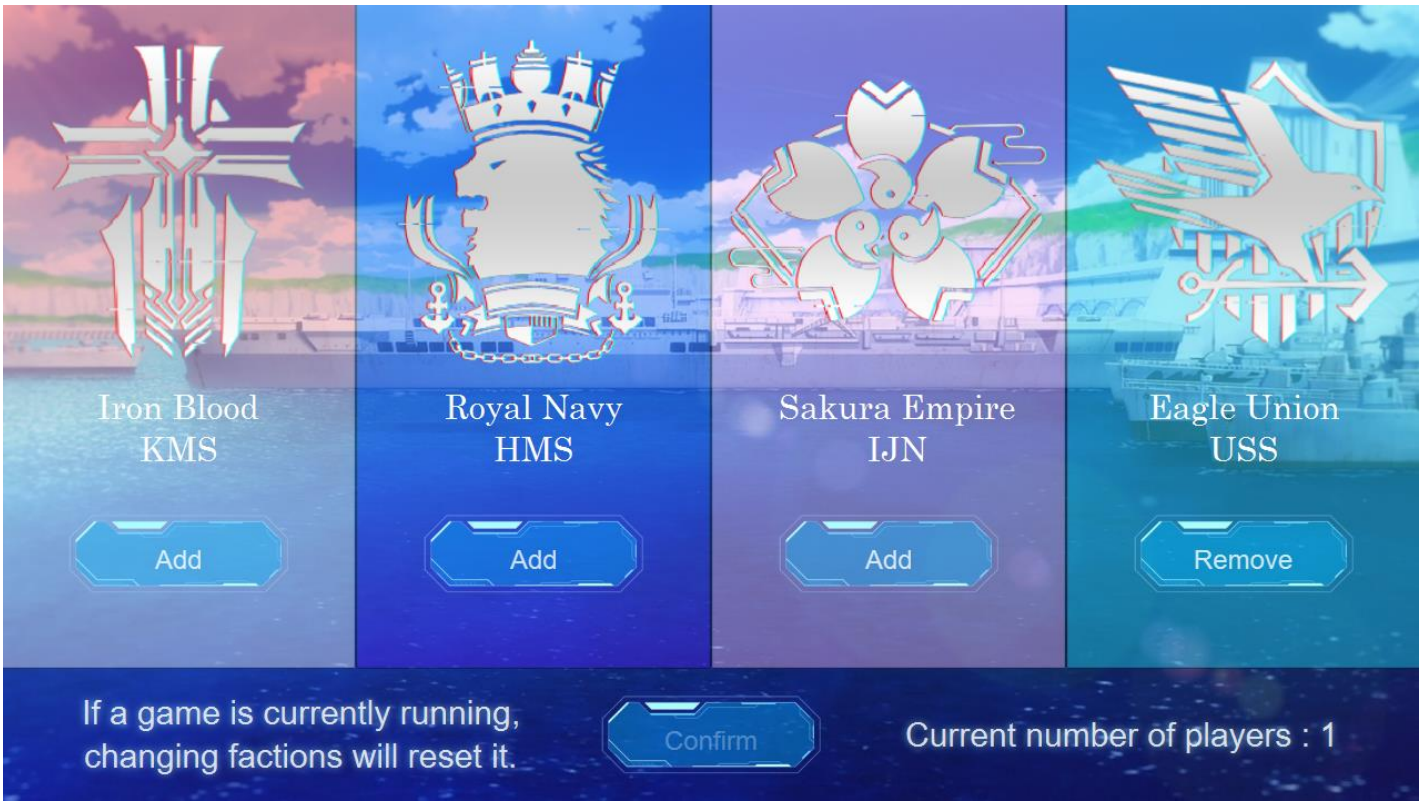

Figure 5: Paramétrage

4 factions navales sont jouables dans ce jeu, chacun possédant des styles différents mais le but principal de cette fenêtre est de **déterminer le nombre de joueur**, qui va du coup jusqu'à 4 maximum. Comme on peut le voir, le jeu ne se laisse pas jouer si le nombre minimum de joueur n'est pas atteint (bouton grisé).

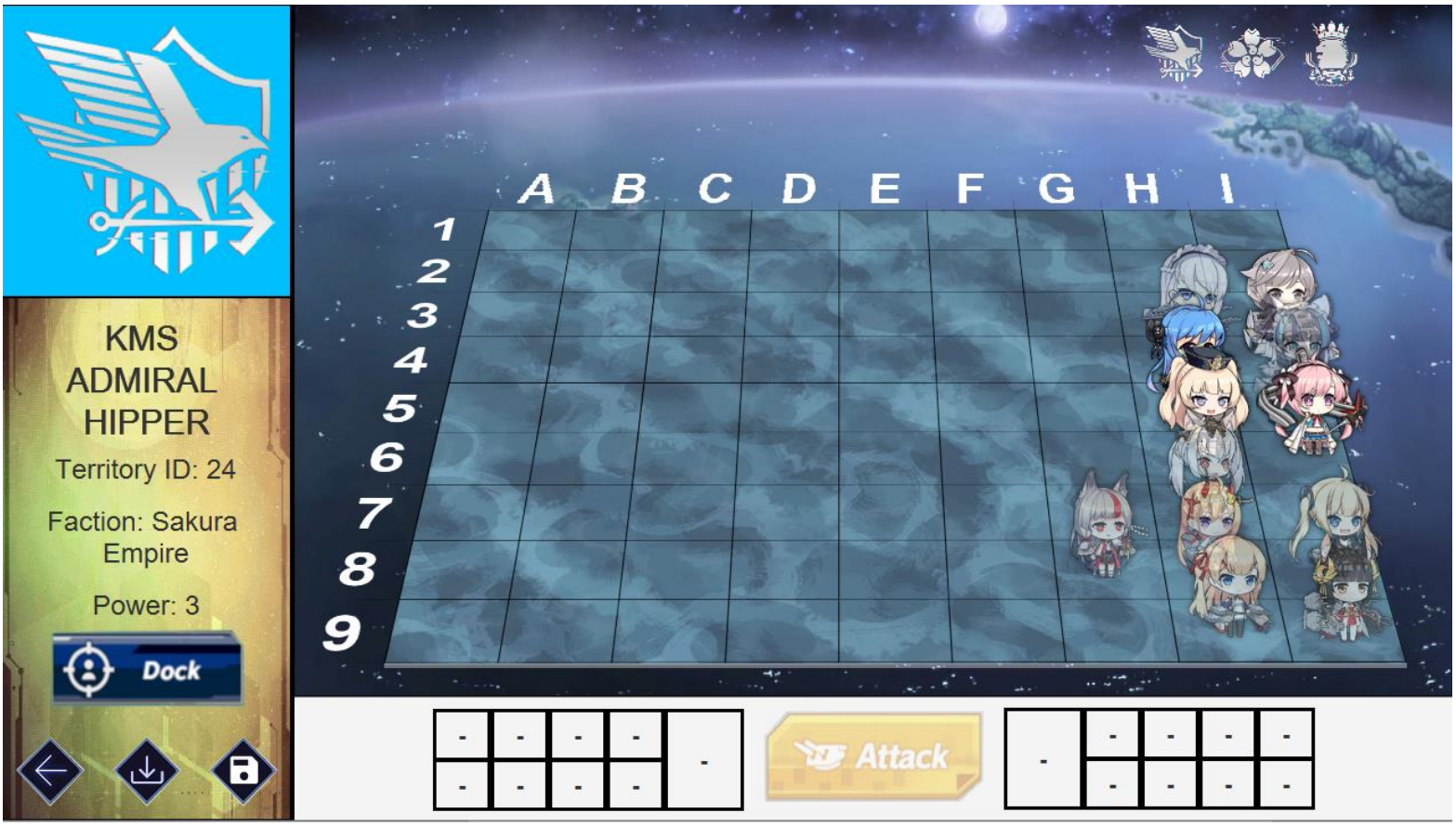

Figure 6: Game mode

Une fois le paramétrage effectué, il suffira d'appuyer sur Play dans le menu principal, ce qui mènera vers cette fenêtre qui est le jeu. Noté tout de même la tentative de ressemblance que nous avons essayé d'établie avec la version réelle du jeu.

Figure 7: Azur Lane

Par exception, nous avons essayé avec 12 navires seulement pour faciliter la compréhension.

Décomposons cette fenêtre en plusieurs partie :

- Tout d'abord, l'affichage du joueur actuel qui joue se fait par le grand symbole de la faction. Ici, par exemple, on sait que c'est le tour du joueur Eagle Union. L'ordre du tour est déterminé aléatoirement, visible en haut à droite.

Les navires du joueurs actuels sont en opacité maximal, tandis que ceux des ennemies sont inférieurs.

Figure 10: Joueur actuel

Figure 8: Ordre des tours

Figure 9: Mise en évidence des navires

Pour savoir les infos d'un navire, il ne s'agit que de passer la souris dessus, là où le menu se mettra à jour, incluant sa faction, sa force, son nom etc. :

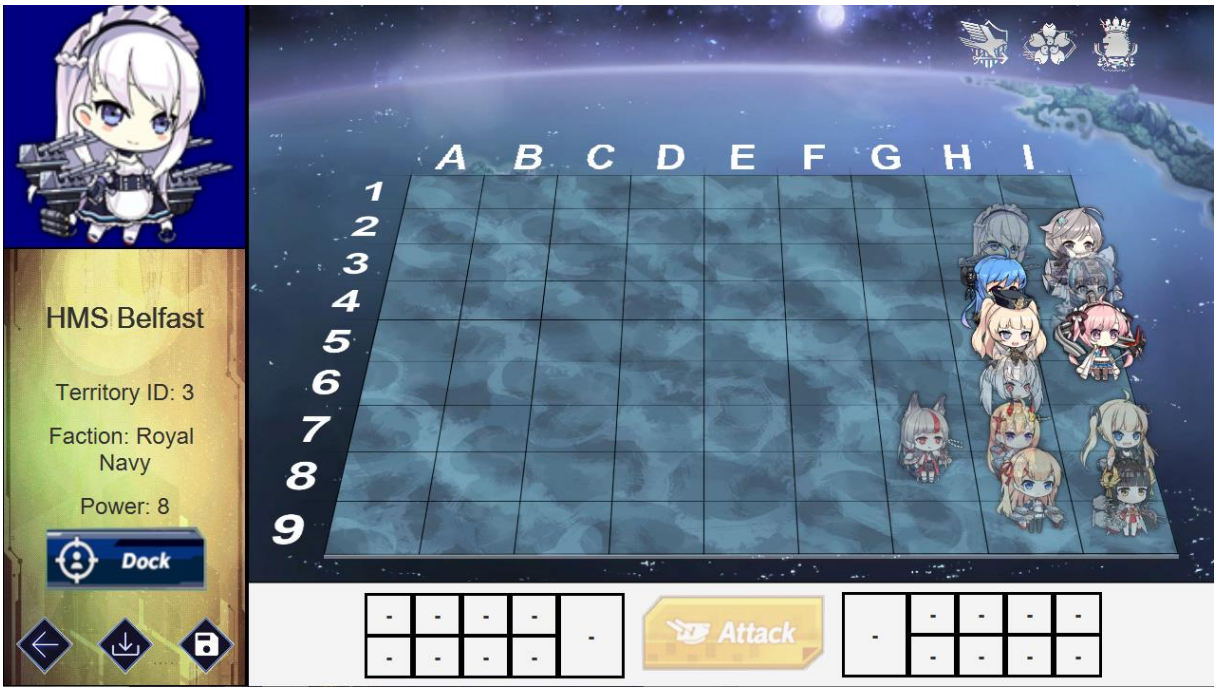

Figure 11: Info d'un navire ennemi

Figure 12: attack disponible

En cliquant sur deux navires (l'un allié, l'autre ennemi voisin), le bouton Attack sera disponible et donnera l'ordre d'attaque, qui simulera notre Version 1, tout comme le reste.

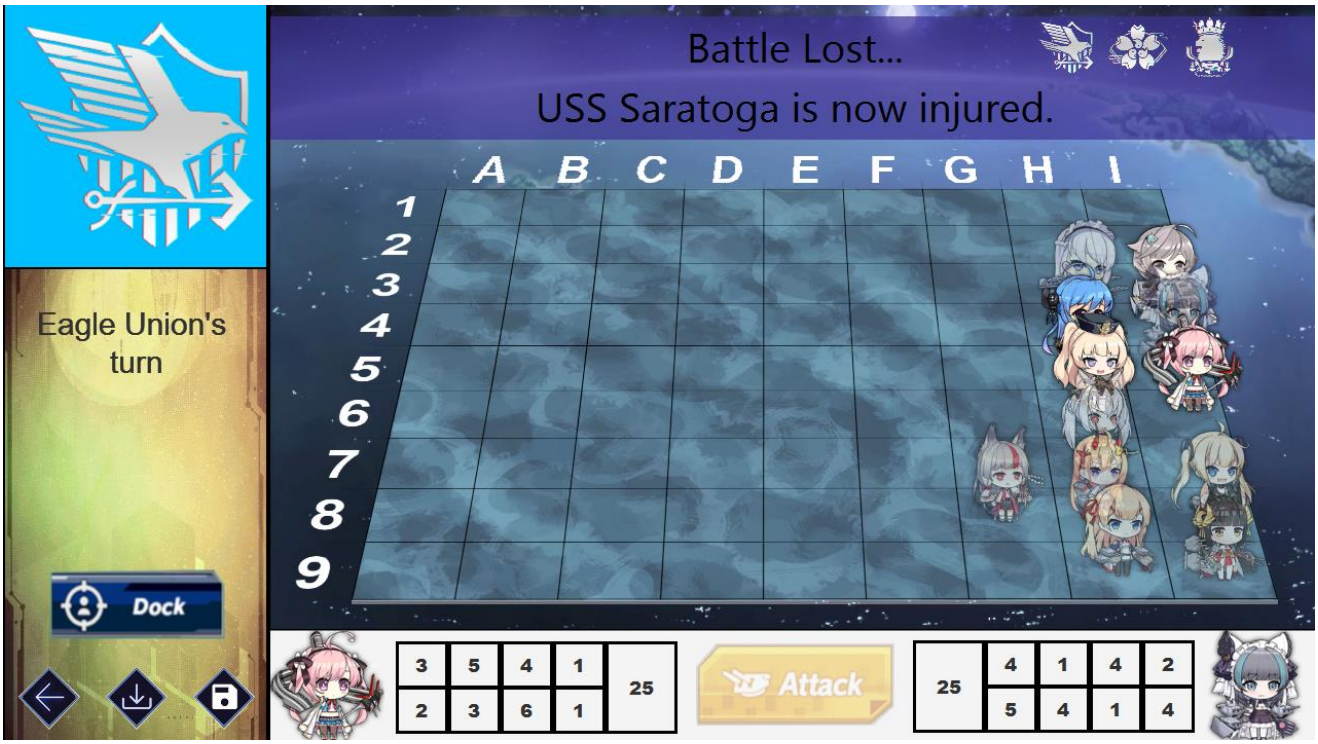

Figure 13: attack échec

IV. Conclusion

Pour conclure, ce projet Java nous a appris beaucoup de choses sur le langage Java, surtout pour l'interface graphique ainsi que les différentes notions abordées en cours. Ce projet était à la fois instructif et amusant à réaliser.