Léo GALMANT
Sacha LENARTOWICZ

# C++ / Mini-projet

# Rapport

## Introduction

Pour ce mini-projet à implémenter en C++, notre tâche était de réaliser une implémentation d'une version basique du jeu Age of War, avec affichage console, en C++. Cette adaptation devait opposer dans un premier temps deux IA en tour par tour. Notre objectif était d'utiliser les différents outils vus en cours et de fournir un programme orienté objet.

## Comment jouer ?

Nous avons fournis dans l'archive un makefile. Il vous suffit donc de vous placer dans le répertoire du jeu, de taper la commande « make » puis la commande « ./ageOfWar ». Vous serez par la suite guidés dans l'utilisation du jeu.

## Fonctionnalités

Lors du démarrage du jeu, nous vous indiquons que deux modes sont disponibles : un mode rapide, qui ne nécessite pas d'appuyer sur « Entrée » pour avancer d'une étape, et un mode plus lent. Dans les deux cas, on vous demande par la suite si vous désirez jouer contre une IA, ou bien laisser deux IA s'affronter.

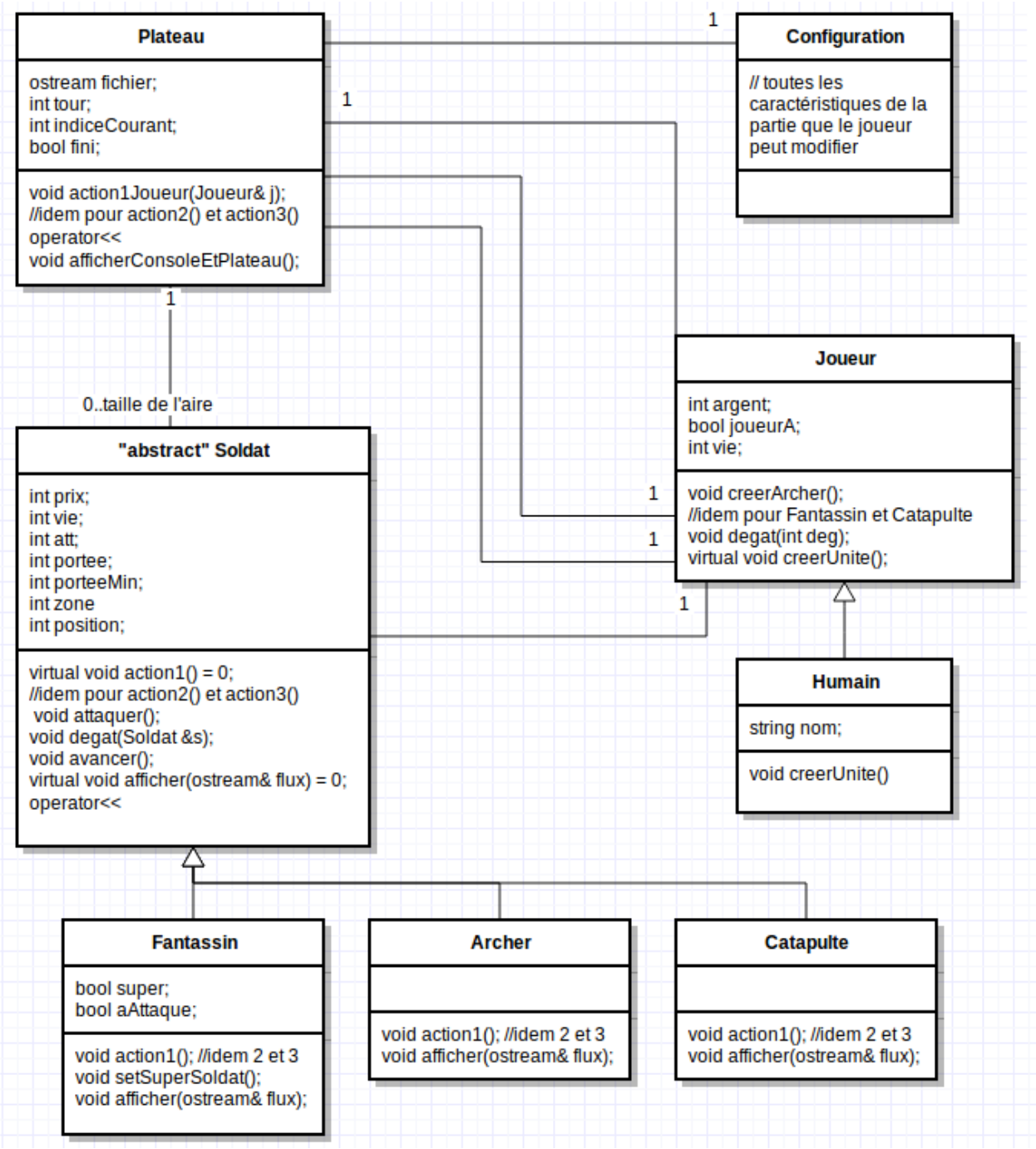
Si vous avez choisi de jouer, vous devez tout d'abord saisir un nom. A chaque tour, les actions de vos Soldats seront effectuées comme le ferait une IA. On vous demandera quel type de Soldat vous voudrez créer (« aucun » faisant partie des réponses possibles), en refusant de créer une unité si celle-ci est trop onéreuse. La partie s'achève lorsqu'un des deux joueurs n'a plus de vie, ou lorsque 100 tours se sont déroulés. Aucun vainqueur n'est déclaré dans ce dernier cas de figure.

Le jeu dispose d'un fichier de configuration qui est à renseigner en paramètre. Celui-ci doit être de la forme de l'exemple donné dans l'archive confStandard.txt. Si l'utilisateur n'entre pas d'argument ou si l'argument est incorrect, le fichier de configuration par défaut, confStandard.txt, est ouvert. Si celui-ci a été supprimé par l'utilisateur, ce sont les valeurs par défaut des variables de

configuration qui sont utilisées. Vous avez à votre disposition un fichier « maConf.txt » que vous pouvez modifier à votre guise.

## Les classes utilisées

Voici le diagramme de classe, avec les principales méthodes utilisées, que nous avons respecté :



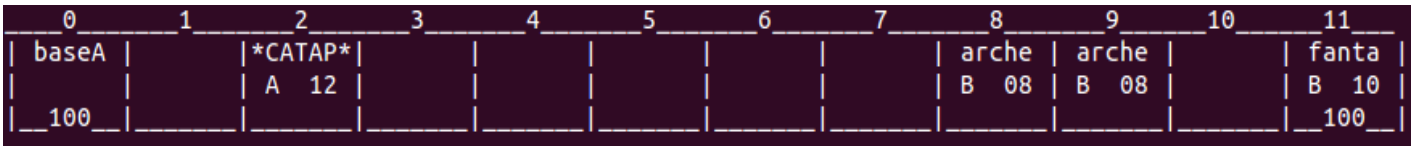
Nous n'avons pas jugé nécessaire de recourir à des templates pour ce projet. En effet ici la programmation objet et l'utilisation de l'héritage a été suffisant pour gérer les collections de soldats. Le plateau est un ensemble de Soldats et les Fantassins, les Archers et les Catapultes étant des Soldats, il n'y a aucun besoin de template.

Comme vous avez pu le constater, la classe Plateau (qui regroupe les éléments de la partie) fournie un élément ofstream appelé « fichier », qui nous permet par la suite d'accéder au fichier « etat_plateau.txt » et de le remplir. Comme les classes Joueur et Soldat propose un lien vers la classe Plateau, ce flux y est alors accessible, et cela nous évite de devoir le passer en argument des fonctions participant au remplissage du fichier annexe.

## L'affichage

Il nous était demandé de pouvoir afficher, à chaque modification de l'aire de jeu, les informations suivantes : numéro du tour, type et position sur l'aire de chaque unité, état des unités et des deux bases, ainsi que le joueur courant, la phase et le marqueur sur l'unité qui joue.

Nous avons donc décidé de vous fournir un affichage suivant ce modèle :




Nous retrouvons ici les différentes positions, numérotées de 0 à 11, en haut de chacune des cases. Lorsqu'un Soldat (une Catapulte, un Fantassin ou encore un Archer) vient d'être créé et occupe une base, l'indicateur « baseA » ou « baseB » est remplacé par le type du Soldat, comme c'est le cas ici en case 11. Le soldat qui voit son type affiché en majuscule et encadré d'étoiles est le joueur qui vient d'effectuer son action. Nous avons préféré cette solution plutôt que de marquer le prochain soldat devant jouer pour une raison de visibilité.

La ligne du milieu des cases occupées par des Soldats indique le joueur du soldat (« A » ou « B ») et sa vie restante. Enfin, la dernière ligne des cases est réservée à l'affichage des points des deux bases (ici, les deux bases ont encore leurs 100 points).

Comme nous devions également afficher le numéro de tour et le joueur courant, nous avons décidé de l'indiquer lors du changement de tour pour le numéro de tour, et lors du passage de main pour le joueur courant. Deux affichages de l'aire de jeu sont obligatoirement séparés par l'action qui a amené au dernier état de l'aire. Nous obtenons ainsi le résultat de l’image page suivante.

Nous devions également afficher ses informations dans un fichier texte, que nous avons appelé « etat_plateau.txt ». Dans ce fichier, nous avons choisi de ne placer que l'action effectuée par un Soldat ou un Joueur, ainsi que les différentes aires de jeu.

```
******************** TOUR 7 ********************



   Au joueur A de jouer. ('s' pour stopper la partie)

       ARCHER : joueur A, pos:4 : avance
____0_____1_____2_____3_____4_____5_____6_____7_____8_____9_____10____11___
| fanta |       |       | arche |       |*ARCHE*|       |       | fanta | catap |       | arche |
| A  10 |       |       | A  08 |       | A  08 |       |       | B  10 | B  12 |       | B  08 |
|__100__|_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|__100__|

       ARCHER : joueur A, pos:3 : avance
____0_____1_____2_____3_____4_____5_____6_____7_____8_____9_____10____11___
| fanta |       |       |       |*ARCHE*| arche |       |       | fanta | catap |       | arche |
| A  10 |       |       |       | A  08 | A  08 |       |       | B  10 | B  12 |       | B  08 |
|__100__|_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|__100__|

       FANTASSIN : joueur A, pos:0 : avance
____0_____1_____2_____3_____4_____5_____6_____7_____8_____9_____10____11___
| baseA |*FANTA*|       |       | arche | arche |       |       | fanta | catap |       | arche |
|       | A  10 |       |       | A  08 | A  08 |       |       | B  10 | B  12 |       | B  08 |
|__100__|_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|__100__|

       Joueur A : creation de catapulte
____0_____1_____2_____3_____4_____5_____6_____7_____8_____9_____10____11___
|*CATAP*| fanta |       |       | arche | arche |       |       | fanta | catap |       | arche |
| A  12 | A  10 |       |       | A  08 | A  08 |       |       | B  10 | B  12 |       | B  08 |
|__100__|_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|__100__|



   Au joueur B de jouer. ('s' pour stopper la partie)

       CATAPULTE : joueur B, pos:9 : attaque ARCHER : joueur A, pos:5
____0_____1_____2_____3_____4_____5_____6_____7_____8_____9_____10____11___
| catap | fanta |       |       | arche | arche |       |       | fanta |*CATAP*|       | arche |
| A  12 | A  10 |       |       | A  08 | A  02 |       |       | B  10 | B  12 |       | B  08 |
|__100__|_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|__100__|
```

Lorsque nous avons implémenté nos différentes, nous avons placé dans chaque destructeur un message indiquant la destruction de l'objet, afin de bien vérifier l'appel. Nous avons par la suite décider de laisser ce message et de l'utiliser pour afficher la mort d'un Soldat à l'utilisateur. Nous obtenons ainsi l'affichage suivant :

```
       CATAPULTE : joueur B, pos:11 : attaque CATAPULTE : joueur A, pos:8
             -->  Destruction de Catapulte
____0_____1_____2_____3_____4_____5_____6_____7_____8_____9_____10____11___
| arche | catap | fanta | arche |       | catap | catap | catap |       |       |       |*CATAP*|
| A  08 | A  12 | A  10 | A  08 |       | A  12 | A  12 | A  12 |       |       |       | B  06 |
|__100__|_______|_______|_______|_______|_______|_______|_______|_______|_______|_______|___94__|
```

## La gestion des attaques

Nous avons décidé de créer une fonction d'attaque générique car le jeu le permettait. Il était effectivement inutile de redéfinir la fonction d'attaque ou de la rendre virtuelle pure car on peut déterminer quelles unités seront attaquées par une unité à partir des éléments suivants :

- le plateau, pour connaître toutes les unités.
- la portée minimum de l'unité attaquante : c'est la distance à partir de laquelle l'unité peut attaquer.
- la portée de l'unité attaquante : la distance maximale à laquelle l'unité peut attaquer.
- la zone d'attaque.
- l'attaque de l'unité attaquante.

Toutes ces données peuvent être retrouvées directement dans la classe Soldat, il on peut donc définir la fonction d'attaque directement dans celle-ci.

## La transformation en super-soldat

Il existe une fonction virtuelle appelée actionMort() dans Soldat, qui prend en paramètre un soldat. Pour l'instant, nous ne l'utilisons que pour le cas du Super-Soldat : à la mort d'un fantassin, on récupère le fantassin qui vient de le tuer puis on le transforme en super-soldat.

Les autres unités du jeu ne redéfinissent pas cette méthode car la mort de celles-ci n'influencent pas pour le moment le comportement des unités du jeu. Toutefois, l'implémentation, par exemple, d'une Super-Catapulte serait facilitée.

## La gestion de l'affichage

Nous avons décidé, pour la classe Soldat, de redéfinir l'opérateur << afin de pouvoir afficher directement un Soldat. Comme l'affichage du type de Soldat va différer pour un Fantassin, un Archer et une Catapulte, nous appelons dans cette redéfinition une méthode afficher(), prenant en paramètre le flux utilisé par l'opérateur << . Ensuite, nous ajoutons au flux des informations sur la position de Soldat et sa vie.

## La gestion de l'IA

Le premier comportement de l'IA que nous avions implémenté consistait à acheter la meilleur unité dès que possible. Cependant, la Catapulte ne pouvais, avec les réglages de base de l'énoncé, qu'être achetée uniquement dans le cas où une unité détruisait une autre, offrant un solde plus important au joueur, ou dans le cas où l'aire de jeu était saturée, obligeant le joueur à économiser.

L'IA que nous vous présentons est un petit peu plus développée. Elle décide, en utilisant une variable aléatoire, si elle doit économiser ou non. Si celle-ci possède assez d'argent pour pouvoir s'offrir une Catapulte (et que la place de la base est libre), alors elle achète forcément une unité (mais pas forcément une Catapulte). Vous pouvez observer ce phénomène en début de partie. Grâce

à cette stratégie, le fait que le joueur A commence la partie ne lui confère plus un gros avantage, puisque les IA adoptent des choix différents.

## Critique du jeu

Nous avons remarqué qu'avec les paramètres de base, l'aire de jeu est très vite saturée. De plus, le joueur qui mène la partie n'a pas à se soucier d'argent, et termine la partie avec souvent plus de 100 pièces d'or. Enfin, toujours avec les paramètres de base, nous avons observé que l'IA qui créait le plus de Catapulte était dans la majorité des cas gagnante. En effet, la meilleure stratégie à adopter lorsque l'on affronte l'IA est d'attendre d'avoir assez de pièces d'or pour pouvoir créer une Catapulte.

## Conclusion

Ce projet nous a permis de travailler un bon nombre de notions étudiées en cours et de se poser la question de quelle solution choisir pour implémenter un problème donné. Nous avons travaillé dans la bonne humeur et la répartition des tâches s'est faite naturellement. Nous sommes tous les deux d'avis qu'une interface graphique peut être intéressante à développer à partir du modèle que nous avons construit.