Projet Wargame

mis en page par
Jouin211, favier211,Ducrot211, Madelaine2??

L1 info ,
Groupe 4A,

29 avril 2022

Sommaire

Table des figures

1 Introduction

1.1 Quel est notre projet ?

Notre projet consiste en la création d'un algorithme d'optimisation pour un WarGame. Le but de ce projet est de fournir un outil permettant d'aider à construire une armée efficace dans une limite de points donnés. Le développement de notre projet s'est séparé en deux étapes.

Tout d'abord, nous avons conçu un moteur de combat entre deux armées avec des règles simples, le but du projet n'étant pas la création d'un simulateur de bataille complexe, mais bien la conception d'algorithmes permettant la génération d'armée.
Dans un second temps, nous avons implémenté plusieurs algorithmes permettant la génération d'armée.

1.2 Pourquoi ce projet ?

Si nous avons choisi ce projet, c'est tout d'abord, car nous trouvions intéressant de comprendre mieux le fonctionnement des algorithmes génétique. De plus, nous savions tous comment fonctionne un wargame, il n'y aurait donc aucune ambiguïté lors de la conception du moteur. Enfin, c'est parce que ce projet nous inspirait énormément, principalement la conception du moteur de combat.

2 Les fonctionnalités implémentées

2.1 Les différentes étapes

Pour réaliser ce projet, nous sommes passés par 4 grandes étapes.

2.1.1 La conception

Pour commencer, nous avons pris le temps de se renseigner afin d'éviter de se retrouver bloqués lors du développement. Ensuite, nous avons construit un diagramme pour nous mettre d'accord sur les objets que nous allons créer et les relations entre ceux-ci. Enfin, à l'aide de notre diagramme, nous nous sommes séparés le travail.

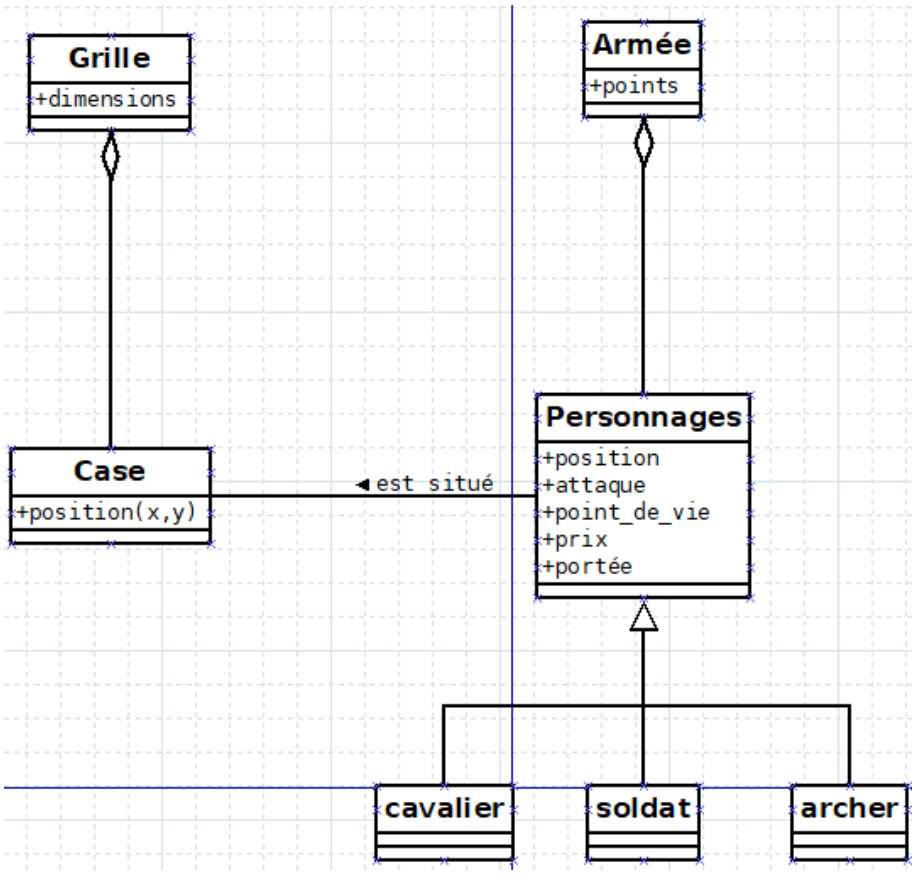

FIGURE 1 – Premier diagramme

Nous pouvons le voir sur la figure 1 que notre diagramme est conçu, le plus simplement possible.

2.1.2 Le développement du simulateur de bataille

Le simulateur de Bataille a été la partie la plus complexe à réaliser, contrairement aux algorithmes génétiques qui ont tous une base similaire, la création du moteur de combat a nécessité de nombreuses heures de tests et d'essais pour créer un moteur compréhensible pour tous les membres du groupe.

2.1.3 Le Développement des générateurs d'armées

Nous avons développé 3 générateurs d'armée, le premier est un générateur d'armée aléatoire, il renvoie une armée générée aléatoirement. il est de loin l'algorithme le plus simple des 3. Le

second est utilise un algorithme d'exploration systématique pour retrouver les meilleures armées. Le dernier utilise un algorithme génétique pour retrouver une des meilleures armées.

2.1.4 Correction des bug et optimisations

Nous avons séparé le travail en plusieurs parties, chacun ayant sa propre manière de programmer, il nous a fallu unifier un minimum le programme. Par exemple en renommant les variables qui portaient à confusion.

2.2 Organisation du travail

2.2.1 répartition du temps

Le diagramme 2 illustre la répartition du temps attribué à chacune des grandes étapes du projet.

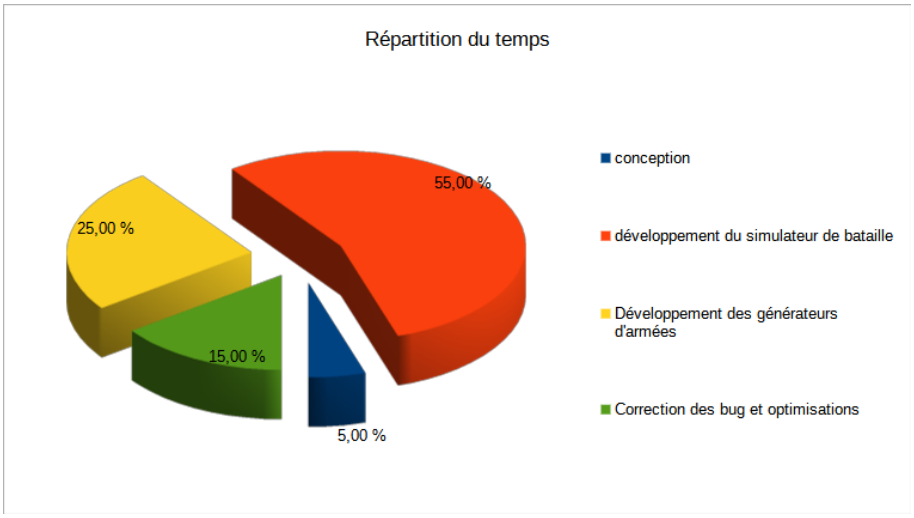

FIGURE 2 – Répartition du temps de travail

Notons que le développement du moteur de wargame a été le plus long. En effet, comme cette tache était la première que nous avions effectuée et comme nous n'avions que peu d'expérience en programmation orienté objet, nous avons pris plus de temps pour créer ce moteur. En second, nous trouvons naturellement le développement des générateurs d'armées.

2.2.2 répartition des taches

Hugo Favier s'est occupé d'une partie de la grille et il s'est aussi occupé de toute l'armée. Il a également travaillait sur l'algorithme génétique et il a aussi participé à la réflexion sur le développement du projet. Il s'est aussi occupé du diaporama et du rapport.

Thomas Jouin s'est occupé d'une petite partie de la grille et aussi fait l'exploration systématique. Il a également optimisé les différentes unités dans le jeux pour par avoir un déséquilibre total entre les unités. Mais il a aussi participé à la réflexion sur le développement du projet. Il s'est aussi occupé du diaporama et du rapport.

Thomas Ducrot n'a été présent qu'à 2 ou 3 séances et n'a fait que participer un peu a la réflexion et fait 2-3 recherches.

Sacha Madelaine n'a été là que durant la première séance donc n'a participé que pour choisir le projet.

2.3 Fonctionnalités

Dans le cadre du projet, nous avons implémenté plusieurs fonctionnalités :

1. la conversion de grille de caractère en grille de personnage. Pour gagner du temps et pour éviter plusieurs problèmes qui pourraient être inattendue, nous avons mis au point la méthode suivante :
   - Avant tout, nous créons une grille composée de chaînes de caractères. il y a un caractère différent pour chaque unité suivie d'un chiffre correspondant à la faction dans laquelle cette unité est, les factions ont pour chiffres 1 et 2, le 0 correspondant à la faction attribuée aux cases vides. Les caractères correspondant aux unités sont les suivants : le "s" pour le soldat, le "a" pour l'archer, le "c" pour le cavalier. Il reste néanmoins la case vide à définir, elle est représentée par un espace (ce qui correspond à " "). Voici comment est définie une grille de taille 4 sur 4 avant que les caractères ne soient convertis.

```
plan = [["s1", "s1", "a1", "c1"],
        [" "," "," "," "],
        [" "," "," "," "],
        ["s2", "c2", "a2", "s2"]]
```

FIGURE 3 – Base pour la construction de la grille

   Notons que bien que l'on puisse définir manuellement une grille, les grilles sont généralement générés automatiquement.
   - Ensuite, nous remplaçons tous les caractères par les personnages correspondants, comme la grille devient illisible à partir de ce moment, nous utiliserons la fonction afficherGrille qui rend lisible la grille remplis d'objet. Nous pouvons voir un exemple de grille de dimension 4 sur 4 ci-dessous.

```
s s a c
- - - -
- - - -
s c a s
```

FIGURE 4 – affichage d'une grille convertie

   Nous pouvons voir dans cette figure que le chiffre correspondant à la faction n'est plus présent, en réalité, les caractères que nous voyons désignent des objets, le chiffre correspondant à la faction est devenu un des attributs de ces objets, il n'est donc plus nécessaire de l'afficher.
2. la génération automatique de grilles. Cette fonctionnalité est utilisée par tous les algorithmes de génération d'armée, elle place deux armées que nous avons données à chaque extrémité d'une grille vide. Il faut cependant savoir que ce générateur de grille ne renvoie qu'une grille de caractères, elle n'est pas utilisable avant d'avoir été convertie en grille de personnages.

3. La gestion des personnages. Chaque personnage peut se déplacer et interagir avec les autres personnages indépendamment. Il y a plusieurs types de fonctionnalités que les personnages peuvent utiliser :
   - Les personnages peuvent utiliser des capteurs pour repérer leurs ennemies, par exemple un personnage peut savoir où est son ennemi le plus proche ou s'il suffisamment proche d'un ennemi pour l'attaquer.
   - Ils peuvent aussi se déplacer sur la grille, le déplacement est assez simple, par exemple si un personnage veut aller dans la case au-dessus de lui, alors il interagit avec la grille pour savoir si la case devant lui est vide, si elle est vide alors ce personnage est dupliqué sur la case correspondante et son ancienne case est remplacée par une case vide.
   - Il y a le combat, qui fait simplement diminuer le nombre de points de vie de l'ennemie ciblée et si l'ennemie n'a plus de points de vie, alors il disparaît.
   - Ensuite, les pénalités diminuent l'attaque et le nombre de points de vie du personnage
   - Enfin il y a la mort du personnage, lorsqu'il n'a plus de point de vie, le personnage se replace par une case vide en utilisant la fonction construitVide qui ne dépend d'aucune classe.

   Ces fonctionnalités représentent l'intégralité des capacités de chaque personnage, en les utilisant toutes, nous obtenons le mouvement qui sera utilisé systématiquement par tous les personnages de la grille lors des simulations de bataille.
4. La simulation de tour. Lorsqu'un tour est simulé, tous les personnages de la grille utilisent le mouvement mentionné plus haut qui consiste en la recherche et l'avancée de notre unité vers l'ennemi le plus proche ou le combat entre le personnage et son ennemi le plus proche s'il est à portée.
5. La simulation de bataille. Cette fonctionnalité est de loin la plus complète de toutes, cette fonctionnalité utilise la simulation de tour cité plus haut en boucle jusqu'à ce qu'il ne reste que les unités d'une même faction sur la grille.
6. La génération d'armée. Nous avons mis au point plusieurs générateurs d'armée différents, ils sont au nombre de 3, il y a :
   - les générateurs basés sur un algorithme génétique et sur un algorithme d'exploration systématique que nous verrons dans les Éléments techniques
   - La génération aléatoire d'armées qui est une fonctionnalité très simple, elle génère une armée aléatoirement. Dans notre programme, elle est utilisée principalement pour générer des armées qui serviront de base à notre algorithme génétique. Cette fonction créée une liste composée de caractères aléatoires entre "s", "a" et "c", ensuite, on l'insère dans notre constructeur.
7. une gestion du prix des armées, cette fonctionnalité est effective si nous donnons une limite de points utilisables par une armée à sa construction. Cette fonction n'empêche pas la construction d'armée dépassant le nombre de points définis, cependant lors des batailles si une armée dépasse son prix limite, alors toutes les unités qui la composent sont alors lourdement pénalisés.

3 Éléments techniques

3.1 Descriptions des différents modules utilisés

Nous avons utilisé plusieurs modules pour faire fonctionner notre programme. Nous avons notamment utilisé le module **copy** pour pouvoir copier les compositions d'armées que nous testons.

Nous avons utilisé le module **random** car on avait besoin pour générer aléatoirement les premières compositions qui vont être testées dans notre algorithme génétique.

3.2 Quelques fonctions très importantes

Nous retrouvons dans notre travail 4 fonctions qui sont au cœur du programme :

1. La fonction **construitPersonnage** est très importante, il s'agit de notre constructeur, celui-ci nous permet de transformer un plan de grille composé de caractères en grille composé de personnages, ce qui permet de simplifier la construction de la grille.
2. Il y a la fonction **trouverEnemie** qui cherche dans la grille tous les personnages qui sont dans la faction ennemie et qui sélectionne le plus proche.
3. Nous avons aussi la fonction **allerEnemie** qui avance d'une case vers l'ennemie sélectionnée(l'ennemie mis en paramètre), si l'ennemie est à portée d'attaque, alors on l'attaque.
4. Et enfin, il y a la fonction **combat**, qui est simplement synthèse de toutes les fonctions utiles au simulateur de combat, cette fonction simule des tours jusqu'à ce qu'une des deux armées ne soit plus présente sur la grille. Lors d'un tour, chaque personnage sur la grille avance vers son ennemi le plus proche et l'attaque s'il est à portée.

3.3 Algorithme génétique et l'exploration systématique

3.3.1 Algorithme génétique

Notre algorithme génétique renvoie une armée ayant gagné un nombre de combat définie par l'utilisateur, cet algorithme génère une unique fois 2 armées aléatoirement. Ensuite, il effectue le test suivant :

il prend ces deux armées et les fait se combattre, un combat est composé en réalité de deux batailles, si la faction 1 gagne alors on ajoute 1 point, si elle perd on lui retire 1 point, on retrouve la faction gagnante en comptant le nombre de points, si celui-ci est supérieur à zéro alors la faction 1 gagne, si ces points sont inférieurs à 0 alors la faction 1 perd, si les deux armées sont à égalité alors l'armée gagnante est choisie aléatoirement, on garde l'armée gagnante et modifie aléatoirement la perdante.

Il répète ce test jusqu'à ce qu'il ait une armée qui a gagné le nombre de victoires défini par l'utilisateur de suite. À ce moment, l'algorithme est terminé, on renvoie l'armée gagnante.

3.3.2 L'exploration systématique

L'exploration systématique a pour but de prendre toutes les possibilités de population et de les faire toute s'affronter, par exemple la première famille affrontera toutes les autres et chaque victoire lui donnera 1 point. On fera à la fin un classement des victoires. Dans notre projet ayant 3 types d'unité différents, nous avons cherché toutes les possibilités en base 3 selon la taille de la grille. Par exemple, si notre grille est de taille 4, nous pourrons avoir des combinaison du

genre 2 1 1 0. Nous convertissons les nombres en lettres selon si c'est un soldat (chiffre 0) ou si c'est un archer (chiffre 1) ou encore un cavalier (chiffre 2). La lettre utilisée étant la première du type d'unité. Ensuite, nous faisons combattre la première combinaison contre toutes les autres et on compte combien de victoire, elle a. On fait ensuite un classement basé sur le nombre de victoires selon les combinaisons. Enfin, on prend la combinaison trouvée par l'algorithme génétique et on regarde en quelle position elle se trouve parmi ce classement.

4 Architecture du projet

Notre projet est organisé entre 3 fichiers principaux et 1 ficher de test. Nous avons donc :

1. Le dossier Armée qui définit les personnages qui seront sur la grille et les Armées. Nous pouvons séparer ce fichier en plusieurs parties.
   - Tout d'abord il y a la classe Armee qui est responsable des modifications et de la gestion des prix des groupes de personnages que nous allons placer sur la grille.
   - Ensuite la classe Personnage, celui-ci est la base des trois unités disponibles. Il est responsable de tous les mouvements des personnages (se déplacer, combattre, rechercher un ennemi ...)
   - Également, il y a les quatre classes Soldat, Archer, Cavalier et Vide qui sont les trois unités et la case vide disponible dans le simulateur, chacune de ces unités hérite des méthodes de la classe Personnage. Il n'y a que les valeurs des attributs qui soient différentes. La classe Vide correspond à un personnage ne faisant pas partie d'une faction et ayant des attributs égaux à zéro.
   - Enfin, le fichier Main appelle tous les autres fichiers pour tester toutes les fonctionnalités du projet

   C'est également dans ce fichier que se situe la fonction construitPersonnage qui ne dépend pas d'une classe.
2. Le dossier Grille qui dépend du dossier armée. En effet, la grille utilise le constructeur qui se situe dans le fichier Armee lorsqu'elle est créée et pour fonctionner. Elle est principalement utilisée pour gérer la grille, mais sert aussi à simuler les batailles notamment avec les fonctions tour et combat
3. Enfin, le dossier baseArmee permet contient les fonctions qui servent à générer les armées, il dépend donc des fichiers vus plus tôt, il définit donc les deux algorithmes vus plus tôt.

5 Démonstration du programme

Dans cette partie, nous verrons un exemple d'utilisation de :

1. Notre simulateur de bataille avec le générateur d'armée aléatoire
2. Notre algorithme de recherche systématique
3. Notre générateur d'armée basé sur un algorithme génétique

Nous montrerons dans cette démonstration chaque étape pour tester chacune de ces fonctionnalités.
Pour garder une taille d'affichage correct, nous prendrons comme exemples uniquement des grilles de taille 3 sur 3 et nous n'afficherons pas le déroulement des combats.

5.1 Utilisation du simulateur de bataille avec le générateur aléatoire d'armée

Tout d'abord, il nous faut deux armées. Nous les générons donc avec le générateur d'armée aléatoire.

```
>>> armee = Armee(200)
>>> armee_exemple_1 = armee.genereLigneAlea(1)
>>> armee_exemple_2 = armee.genereLigneAlea(2)
>>> armee_exemple_1
['c1', 's1', 'a1']
>>> armee_exemple_2
['c2', 's2', 'c2']
>>>
```

FIGURE 5 – génération de deux armées aléatoirement

Ensuite, nous plaçons ces deux armées sur la grille et nous lançons une bataille. Lorsque nous lançons le combat, nous mettons en paramètre 0 pour éviter d'afficher graphiquement le déroulement du combat. Pour l'afficher, nous devrions remplacer ce paramètre par 1.

```
>>> plan = armee.placerArmees(armee_exemple_1,armee_exemple_2)
>>> grille = Grille(plan)
>>> grille.afficherGrille()
c s a
- - -
c s c
>>> grille.combat(0)
1
>>>
```

FIGURE 6 – placement des armées, affichage de la grille et simulation d'un combat

Comme nous n'affichons pas le déroulement des combats, nous pouvons voir que le résultat du combat est directement affiché, dans notre cas, le chiffre 1 correspond à une victoire de la première armée.

5.2 Utilisation de l'algorithme de recherche systématique

Pour le bon fonctionnement de notre algorithme de recherche systématique, nous avons besoin de connaître les règles qui encadre les armées, il nous faut donc un objet Armee. Nous pouvons voir sur la figure 7 comment nous procédons.

```
>>> armee=Armee(200)
>>> genereClassement(armee)
```

FIGURE 7 – création d'un classement

Le résultat de ces commandes correspond à une liste d'armées triée par puissance, chaque armée de cette liste est représenté par son nombre de victoires sur tous les combats effectués dans l'algorithme de recherche systématique. Cette liste étant composée de 27 armées, nous ne l'afficherons pas.

5.3 utilisation d'un générateur d'armée basé sur un algorithme génétique

Comme pour notre algorithme de recherche systématique, nous avons besoin de connaître les règles qui encadrent les armées, nous créons donc un objet Armee et lançon notre algorithme avec cet objet comme paramètre. Nous pouvons voir sur la figure 8 comment nous procédons.

```
>>> armee=Armee(200)
>>> genereArmees(armee)
['c', 's', 's']
>>>
```

FIGURE 8 – Génération d'une armée avec l'algorithme génétique

Nous avons obtenu un résultat, nous pouvons donc chercher sa position dans le classement qui a été produit par l'algorithme de recherche systématique pour savoir si l'algorithme génétique s'est montré efficace.

```
>>> classement = genereClassement(armee,3)
>>> classementComposition(classement,['c','s','s'])
2
>>>
```

FIGURE 9 – comparaison entre nos deux algorithmes

Comme nous pouvons le constater sur la figure 9 notre algorithme génétique à sélectionné l'armée classée seconde dans le classement, il n'est donc pas aussi efficace que notre algorithme de recherche systématique (l'armée qu'il a crée n'est pas la meilleure du classement.), cependant, il reste très efficace et est bien plus rapide que notre algorithme de recherche systématique, cette différence est bien plus présente sur des grilles de plus grandes tailles.

6 Conclusion

Nous avons tout au long du projet gagné beaucoup d'expérience grâce aux nombreuses recherches et découvertes que nous avons réalisées. Nous avons remarqué que le travail de groupe lors d'un projet n'est pas aussi simple que cela puisse paraître puisqu'il est difficile de savoir comment répartir le travail. En raison de l'absence des 2 autres membres du groupe, notre organisation s'est simplifiée. Nous sommes contents d'avoir choisi et réalisé ce projet bien qu'il n'ait pas atteint tous les objectifs que nous avions imaginés au début. Effectivement, nous avions mis la barre trop haute, nous nous en sommes rendu compte grâce à l'intervention du professeur. Lors de la conception, la partie qui nous a pris le plus de temps est la réalisation du moteur du jeu en raison de sa complexité. Puis nous avons fait l'exploration systématique ainsi que l'algorithme génétique assez facilement et rapidement. En raison d'un manque de temps, nous n'avons pas pu améliorer notre algorithme génétique comme nous le souhaitions. En effet, notre but, concernant l'algorithme génétique, était de pouvoir tester toute une population et non pas deux individus à la fois, cela nous aurait permis d'accélérer notre algorithme. Durant le projet, beaucoup de possibilités d'améliorations nous sont venues en tête, mais nous ne les avons pas réalisées en raison du peu de temps que nous avions.
Voici la liste des améliorations que nous aurions aimé réaliser :

- Ajouter plusieurs nouvelles unités
- Ajouter un système de moral sur les armées
- pouvoir imposer au générateur d'armée des règles pour le rendre plus efficace, par exemple, éviter de mettre un archer en première ligne
- les soldats deviennent moins fort après qu'un combat soit finis en raison de l'épuisement
- Ajouter un leader qui donne différents bonus

On peut aussi mentionner notre manque d'expérience au début du projet et donc du manque d'efficacité des premières fonctions qu'on a fait notamment par rapport à la grille.