

Année universitaire 2016-2017
Université de Caen Basse-Normandie

Rapport de Projet : Optimiseur de Wargame

CANUET Maxime, CORBET Thorvald,
JOLIVEL Valentin, L'HOMME
Florentin
L2 Informatique

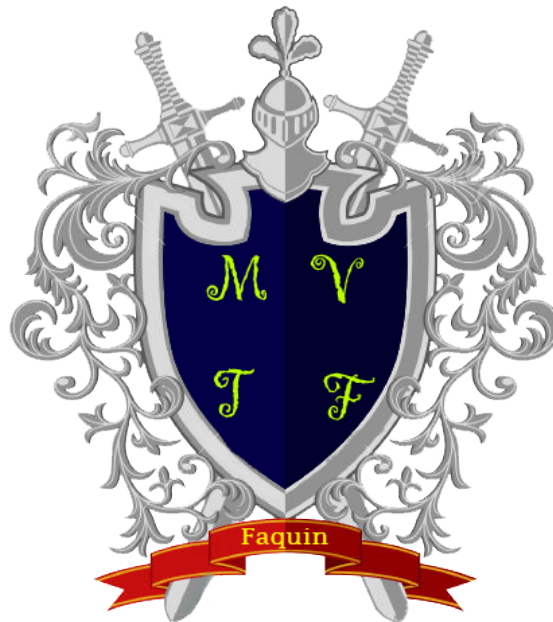


Table des matières

1	Présentation du jeu de Wargame	3
1.1	Introduction	3
1.2	Qu'est-ce qu'un Wargame ?	3
1.3	Les règles du Wargame	3
1.4	Comment jouer au Wargame	3
1.4.1	Phase de préparation	3
1.4.2	Phase de jeu	3
2	La réalisation du jeu en mode console	4
2.1	Les plateaux	4
2.2	La création des unités	4
2.2.1	Le calcul de points	4
2.3	Le csv	5
2.3.1	Les unités dans le programme	5
2.4	La création des armées	6
2.5	Algorithme Astar	6
2.5.1	Pré-requis	6
2.5.2	Principes de l'algorithme et heuristique choisie	7
2.6	Le déplacement des armées	8
2.7	la phase d'attaque	8
3	Les Intelligences artificielle	8
3.1	L'IA simple	9
3.1.1	La création d'armée via l'IA	9
3.1.2	Le placement des unités	9
3.1.3	Le déplacement	9
3.1.4	Les attaques	9
3.2	L'IA médium	9
4	L'optimiseur	10
4.1	Algorithme génétique	10
4.1.1	le programme	10
5	La réalisation du jeu en mode graphique	10
5.1	La charte graphique	11
5.2	Préparation du combat	11
5.2.1	Explication du code	12
5.3	Phase de création de son armée	12
5.4	Phase de jeux :	13
5.4.1	Explication du code	13
6	Tournois	15
7	Algorithme génétique	15

8	La sortie web	16
8.1	Introduction	16
8.2	Disposition de la page web :	16
8.2.1	Affichage de données générale	16
8.3	résultats de l'exploitation des données	17
8.3.1	Utilisation d'armées aléatoire uniquement	17
8.3.2	Ajout des armées pur	18
8.3.3	Conclusion	18
9	Conclusion	19
10	Les problèmes rencontrés	19
11	Annexe	20
11.1	Vocabulaire	20
11.2	Wiki	20
11.3	Les ressources d'informations	20
11.4	Les outils utilisés	20

1 Présentation du jeu de Wargame

1.1 Introduction

1.2 Qu'est-ce qu'un Wargame ?

Un Wargame est un jeu où le joueur est plongé dans une situation de conflit, une guerre, une escarmouche ou tout autre environnement se prêtant à des batailles. La célèbre série de jeu Starcraft est un bon exemple de Wargame. Pour notre projet nous avons décidé de réaliser un Wargame dans un monde avec un style Héroïque fantaisie.

1.3 Les règles du Wargame

Notre Wargame est un jeu de type tour par tour, la seule condition pour remporter la victoire est l'annihilation de l'équipe adverse (L'annihilation signifie que plus aucune unité adverse ne doit être présente sur le plateau de jeu). Chaque tour est divisé en deux phases bien distinctes. La première est la phase de déplacement. La seconde est la phase d'attaque. Pour de plus ample informations, consultez le livre de règles. Enfin chaque armée possède un nombre égal de points à répartir dans l'achat de ses unités (ce nombre est égal à 1000).

1.4 Comment jouer au Wargame

Notre Wargame est divisé en plusieurs phases que nous allons détailler ci-dessous.

1.4.1 Phase de préparation

Notre jeu est jouable en mode graphique. Pour pouvoir y jouer, lancez le fichier mainGUI.py. Une fois cela fait, cliquez sur le bouton "Jouer". Sélectionnez ensuite le type de joueur que vous désirez (Humain ou alors l'une des différentes IA proposées). Une fois cette action effectuée, si vous aviez décidé de jouer avec un joueur humain vous avez la possibilité de choisir son armée parmi celles présentées (Pour le moment, seul les Humains ainsi que les Orks possèdent des icônes pour leurs troupes). Dès que votre choix est fait, le jeu vous proposera de créer votre armée en choisissant parmi 7 unités différentes. Cliquez ensuite sur le bouton "Stop" pour lancer la partie.

1.4.2 Phase de jeu

Une fois votre armée prête, la phase de jeu débute. Comme nous l'avons dit précédemment, le jeu se déroule en tour par tour de ce fait le premier joueur commence par placer ses unités en cliquant sur la grille. Une sécurité est mise en place afin que le joueur ne puisse pas poster ses troupes n'importe où, la zone de départ étant de 4 cases de hauteur et aucune restriction sur la largeur. Une fois que les deux joueurs ont placé leurs troupes, la bataille peut commencer !

Pour cela les joueurs commencent par déplacer leurs troupes sur le terrain en cliquant sur l'unité qu'ils souhaitent déplacer puis sur la case de destination. Bien évidemment des mesures ont été prises pour cadrer les actions du joueurs, il ne peut par exemple pas déplacer une unité plus que ses points de déplacement ne le peuvent. Suit alors la phase d'attaque qui se déroule comme la phase de déplacement sauf que l'on ne clique pas sur une case vide mais une case contenant un ennemi. Un message sera affiché au joueur pour savoir si l'unité peut attaquer ou non (attaque déjà faite ou pas d'ennemis à portée). Une fois qu'une des deux équipes est complètement éliminées, un message de victoire apparaît signalant le joueur vainqueur.

2 La réalisation du jeu en mode console

Après réflexion le groupe s'est accordé sur le fait de créer le jeu en mode console dans un premier temps. En effet pour la réalisation de l'IA et pour le bon fonctionnement de l'algorithme génétique le jeu doit être parfaitement jouable en mode console.

2.1 Les plateaux

Dans notre jeu nous avons défini deux plateaux bien distinct. Un premier plateau dans lequel nous allons placer les unités des deux armées et un deuxième contenant l'environnement (Éléments naturels et constructions humaines). Ce second plateau ne contient actuellement que les valeurs 0 et 1 pouvant être respectivement traduite par "plaine" et "point d'eau", autrement dit une zone accessible aux unités et une zone non traversable. Une méthode de l'objet Plateau permet de générer une grille d'obstacles à partir de ces deux grilles (Notamment utilisée pour l'algorithme A*).

2.2 La création des unités

Dans un premier temps nous avons créé les unités dans un tableau pour pouvoir définir les statistiques de chacune d'elles et mettre en place un calcul de points (coût) par unité. Cette représentation sous forme de tableau nous donnait un bon aperçu des données de chaque armée. Nous avons donc pu, par exemple, équilibrer le prix des unités pour obtenir des armées assez équilibrées (En prenant en compte les spécificités de chacune, par exemple les Orks sont moins puissant mais plus nombreux).

2.2.1 Le calcul de points

Pour calculer le prix de chaque unités des différentes armées, nous nous sommes basé sur une formule prenant en compte toutes les statistiques. Ce calcul est le suivant :

Prix : Potentiel offensif + Potentiel défensif + portée

HUMAINS	PV	ARMURE	PRECISION	ATQ	DÉPLACEMENT	PORTÉE	TAUX D'ESQUIV
ENROLÉ	10	2	50	10	6	1	1
ÉPÉISTE	25	10	60	20	5	1	5
LANCIER	25	10	60	20	5	2	2
CHEVALIER	40	30	70	40	3	1	1
ARCHER	15	5	90	20	6	5	7
ARBALÉTRIER	20	7	90	25	6	6	4
CAVALIER	25	10	60	20	7	1	2
GÉNÉRAL	80	20	80	50	7	1	2

FIGURE 1 – Exemple : Tableau des unités de l'armée "Humains"

Potentiel offensif : $\lfloor \text{attaque} * (\text{precision} + \text{taux de critique} + 1) \rfloor / 100$

Potentiel défensif : $\lfloor (\text{PV} + \text{armure}) + ((\text{PV} + \text{armure}) * \text{taux d'esquive}) / 100 \rfloor$

portée : déplacement + portée

2.3 Le csv

Nous avons utilisé le tableur pour entrer toutes les informations relatives aux statistiques des unités des différentes armées. Cependant, pour réutiliser ces données en python nous avons utilisé des fichiers .csv créés directement depuis le tableur. Cela permet une réutilisation simple des statistiques de base des unités, en allant chercher les informations dans le csv, et cela permet aussi de pouvoir modifier de façon claire les unités à partir du tableur (il suffit ensuite de produire de nouveaux fichiers .csv) tout en pouvant faire divers calculs sur ces statistiques. Les fichiers csv contiennent toutes les informations du tableur, les lignes les unes en dessous des autres et les colonnes séparées par des virgules.

2.3.1 Les unités dans le programme

Les unités sont définies dans le fichier *"unite.py"*. Dans la classe nommée *"Unite"* on définit toutes les caractéristiques de chaque unité. En plus de cela, nous ajoutons ses coordonnées, la quantité de déplacement restante (En case) ainsi que le nombre d'attaques pouvant être encore effectuées. Les coordonnées renseignées permettent un déplacement simplifié mais permettent aussi de détecter plus rapidement l'ennemi le plus proche, ce en analysant les coordonnées des ennemis via la composition de l'armée plutôt qu'en analysant l'ensemble du plateau de jeu. Nous avons ensuite créé une *"factory"*. La *"factory"* est une méthode abstraite qui crée une unité à partir de sa race et son nom en lui appliquant ses caractéristiques. Enfin la *"factory"* est utilisée dans la fonction *"addUnite"* pour pouvoir créer une armée et ce en tenant compte du solde de point encore restant.

2.4 La création des armées

Pour créer son armée, le joueur doit cliquer sur les symboles "+" pour ajouter une unité et "-" pour la supprimer. Il répète cette action jusqu'à atteindre le seuil pré-fixé.

2.5 Algorithme Astar

À de multiples reprises, cet algorithme se révèle très utile. Astar, permettant de trouver l'un des plus courts chemins entre un point D (départ) et un point A (arrivée), se voit utile tout d'abord en mode joueur humain. Le joueur cherchant à déplacer ses unités au travers de la carte, l'application se devait d'imposer un système "réaliste" de déplacement, c'est à dire sans traverser un obstacle tel qu'un point d'eau, une construction (Ex : une maison) ou encore des régiments ennemis (Traverser un régiment allié peut être toléré). En prenant en compte des obstacles, comme ceux précédemment cités, Astar renverra donc un chemin s'il en existe un. Il ne restera plus qu'à vérifier si l'unité que l'on souhaite déplacer possède assez de points de déplacements pour emprunter ce chemin. En plus du mode joueur humain, Astar trouve une grande utilité du côté du mode joueur IA jusqu'à en devenir presque indispensable. En effet, en mode joueur humain, nous aurions pu faire abstraction de cet algorithme en demandant simplement au joueur de "tracer" lui même le chemin à traverser, par la suite, il aurait été facile de vérifier la validité de ce chemin (Obstacles, longueur etc...). En mode joueur IA vient alors un problème : Comment faire en sorte que l'IA choisisse un chemin plus ou moins optimisé (Court et sans obstacles) ? Là intervient Astar, une fois choisi un point d'arrivée, l'algorithme remplit sa fonction en renvoyant un chemin allant des coordonnées initiales de l'unité jusqu'à sa destination. Contrairement au cas d'un joueur humain, nous ne sommes pas obligé de refuser un déplacement trop long proposé par Astar, au contraire, grâce aux coordonnées dans le plateau de jeu, on peut déterminer quelle unité ennemi est la plus proche (par exemple avec les distances de manhattan ou encore en lançant un Astar vers plusieurs unités ennemis) et prendre comme destination ses coordonnées. Si Astar renvoie un chemin, c'est qu'il en existe un jusqu'à la cible, on peut alors diriger notre unité vers celle-ci en parcourant, même partiellement, le chemin obtenu via l'algorithme. En d'autres mots, les déplacements d'un joueur IA repose essentiellement sur l'Algorithme Astar.

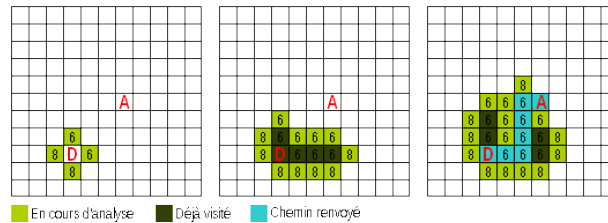
2.5.1 Pré-requis

L'exécution de l'Algorithme Astar nécessite bien évidemment quelques pré-requis (arguments à renseigner dans l'appel de la fonction), de façon évidente les coordonnées initiales et les coordonnées finales espérées ainsi qu'une grille composée uniquement de 0 (Obstacles) et de 1 (Cases libres). Le plateau de jeu ne peut donc être fourni de façon brut, il faudra alors dans un premier temps réaliser une grille correspondante à l'analyse de notre plateau de jeu, ceci

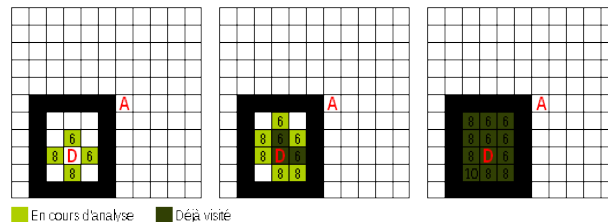
étant réalisable via la méthode grilleObstacle de l'objet Plateau. Optionnellement, nous pouvons indiquer à Astar la liste des directions dans lesquelles le déplacement est possible, par défaut direction = [(1,0), (-1,0), (0,1), (0,-1)] soit "Droite", "Gauche", "Bas" et "Haut".

2.5.2 Principes de l'algorithme et heuristique choisie

Initialement, Astar commence par les coordonnées de départ renseignées. À partir de celles-ci, on analyse les successeurs possibles, autrement dit les cases vers lesquelles nous pouvons nous diriger. On attribue alors à ces cases une heuristique, un coût théorique/estimé depuis le point de départ jusqu'à l'arrivée. L'heuristique choisie correspond au nombre de déplacements effectués depuis le départ + la distance de manhattan entre la position actuelle et l'arrivée. Dans l'exemple ci-dessous, chacune des cases en cours d'analyse (Première image à gauche) ont effectué 1 déplacement et + 5 de distance de manhattan pour celles situées en haut et à droite ou + 7 pour celles situées en bas et à gauche. Deux des cases ont donc une heuristique égale à 6, les deux autres égale à 8.

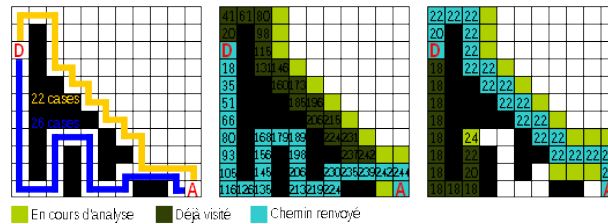


On continue alors l'algorithme en prenant la ou l'une des heuristiques les plus faible, notre but étant de trouver un chemin court de préférence. On prendra donc ici l'une des cases d'heuristique 6, on analyse ensuite les successeurs possibles non visités, la case de départ étant notre point d'origine fait donc évidemment parti des cases déjà visitées. On calcule les heuristiques de ces successeurs et on recommence en prenant l'une des cases avec le minimum d'heuristique, ce jusqu'à parvenir au point d'arrivée ou, dans le cas contraire, ne plus avoir de cases à visiter. Une fois terminé, Astar renvoie une liste de coordonnées s'il trouve un chemin valide entre le départ et l'arrivée, s'il ne trouve pas de chemin, il renvoie False.



Pour obtenir de bons résultats avec Astar, il est nécessaire de choisir une

bonne heuristique. Pour le schéma présenté ci-dessous, le chemin jaune est le plus court des deux présentés, au centre se trouve le résultat obtenu avec une mauvaise heuristique, à droite celle semblant être la meilleure et, accessoirement, étant celle que nous avons retenue. Au centre, l'heuristique correspond à la somme de l'heuristique précédente (0 pour le point de départ) + 1 (Le déplacement) + la distance de manhattan jusqu'à l'arrivée, avec cette heuristique, nous n'obtenons visiblement pas le meilleur résultat. A droite, l'heuristique correspond à la somme de la distance parcourue depuis le point de départ (Distance réelle et non de manhattan) + la distance de manhattan jusqu'à l'arrivée.



2.6 Le déplacement des armées

Le déplacement des armées se fait unité par unité. Notre fonction "*deplacementArmee*" permet au joueur d'entrer la case sur laquelle l'unité veut se déplacer. Elle vérifie ensuite si cette case est contenue dans la zone de déplacement possible. Enfin, elle vérifie que la case soit vide pour ensuite pouvoir effectuer le déplacement.

2.7 la phase d'attaque

La phase d'attaque est permise aux unités qui ont au moins un ennemi à portée. La fonction qui s'occupe de l'attaque pour un joueur humain se nomme "attaquer", celle-ci fait appel à d'autres fonctions comme "*testPrecision*" qui va déterminer si l'unité réussit son attaque, une autre permet de tester si l'ennemi esquive le coup, une autre encore supprime des points de vie à une unité touchée.

3 Les Intelligences artificielle

Nous voulons proposer différentes IA pour permettre aux joueurs d'éprouver leurs facultés de stratégies contre des IA plus ou moins intelligentes. Pour la première partie de notre projet nous nous sommes contentés de deux IA. Une première, qui laisse découvrir aux joueurs les mécanismes du jeu. Et une seconde, permettant aux joueurs de jouer pour se détendre. Ces deux IA sont prévues pour nous donner une base et permettre aux joueurs de s'habituer au jeu. Par la suite nous voudrions développer une IA capable de donner du fil à retordre aux utilisateurs.

3.1 L'IA simple

L'IA simple est une IA très rudimentaire. Elle sert de base pour les autres IA (Elle sera aussi utilisée dans l'algorithme génétique à titre d'exemple, mais pas pour obtenir une véritable armée optimisée). Comme les autres IA, celle-ci possède son propre fichier permettant de la sélectionner pour le déroulement d'un combat. Nous allons vous détailler son fonctionnement.

3.1.1 La création d'armée via l'IA

L'IA simple se contente d'aller rechercher des armées pré-faites dans un fichier csv. L'IA va chercher la race ainsi que la composition d'une armée pré-enregistrée dans le csv.

3.1.2 Le placement des unités

Cette IA place ses unités de manière aléatoire sur le plateau en respectant la zone de placement. Pour chaque unité, elle vérifie la validité du positionnement, autrement dit elle vérifie que les coordonnées choisies ne correspondent ni à une unité déjà placées ni à un obstacle naturel. Si les conditions sont respectées, l'unité est placée, sinon de nouvelles coordonnées sont (re)tirées aléatoirement.

3.1.3 Le déplacement

Cette IA se déplace de manière aléatoire sur le plateau et attaque si un ennemi se trouve à portée. Son fonctionnement est relativement simple, en effet on parcourt la liste des unités de notre armée puis on parcourt celle de l'ennemi. Les unités potentiellement attaquable sont placées dans une liste qui sera utilisée pour l'attaque. Si la liste est vide, on crée une liste dans laquelle seront mises les cases possible pour le déplacement de l'unité. Enfin, le déplacement est choisis au hasard dans cette dernière liste.

3.1.4 Les attaques

La fonction d'attaque pour l'IA reprend le même principe que la fonction d'attaque pour un joueur humain à ceci près que, si l'unité de l'IA a à portée plusieurs unités ennemis, elle se dirige vers le plus proche.

3.2 L'IA médium

Cette IA est encore en développement, elle aura des déplacements rudimentaires. En effet chaque unité se déplacera vers l'unité ennemie la plus proche.

4 L'optimiseur

4.1 Algorithme génétique

L'algorithme génétique est un objet qui permet à d'effectuer un certain nombre de combats et de sélectionner les armées les plus optimisées pour le jeu. Pour cela on donne à l'algorithme le nombre de tours que l'algorithme devra effectuer, le nombre d'armées qui seront utilisées au départ de l'algorithme, le pourcentage d'armées qui resteront inchangées, le pourcentage d'armées qui seront mutées, le pourcentage d'armées qui seront croisées, et un paramètre qui indique si des armées 'pures' seront utilisées durant le premier tour de l'algorithme. On entend par armées 'pures' des armées constituées d'un seul type d'unités.

L'algorithme commence donc par générer le nombre requis d'armées : suivant les paramètres donnés soit les armées sont générées de manière totalement aléatoires, soit on y ajoute les armées 'pures'.

Ensuite l'algorithme utilise la fonction tournois pour faire s'affronter les armées et déterminer lesquels sont les meilleures. Une fois le classement effectué l'algorithme met dans une nouvelle liste d'armées les meilleurs armées du tournois qui restent inchangées. L'algorithme procède ensuite à des mutations sur certaines armées et produit de nouvelles armées en effectuant des croisements entre d'autres armées. Une fois tout ceci effectué l'algorithme boucle jusqu'à ce qu'il ait effectué le nombre d'itération requis. Après cela l'algorithme sauvegarde les informations concernant les mutations et les croisements qu'il a effectués et affiche le temps qu'il a mis à s'exécuter.

4.1.1 le programme

Le système de tournois mis en place est relativement simple. Une population d'armée est renseignée à la fonction donc, pour une population de N armées, il y aura $(N^2 - N)/2$ combats différents (Combat avec soi-même exclu). Chaque cas est réalisé par une simple double-boucle. Néanmoins, le hasard peut parfois grandement intervertir le résultat d'un combat, pour éviter donc ces cas, un combat entre une armée N_1 et une armée N_2 est rejoué plusieurs fois, celle qui remporte le plus de victoires est donc considérée comme la gagnante de l'affrontement. Les victoires des affrontements sont ensuite comptabilisées ce qui, à la fin, génère un classement. Il ne reste plus qu'à sélectionner un certain nombre d'armées parmi les plus fortes pour continuer dans l'algorithme génétique.

5 La réalisation du jeu en mode graphique

La partie graphique du jeu a été réalisé avec Pygame. Le mode graphique se découpe en deux parties, la première partie est le fichier `mainGUI.py`, c'est ce fichier qu'il faut ouvrir pour lancer le jeu en mode graphique. Ce fichier contient peu de choses, il permet juste le lancement des fonctions permettant l'affichage des fenêtres de jeux. La seconde partie est le fichier `GUIfunc.py`, c'est

ce fichier qui est le cœur du mode graphique, chacune des fonctions de ce fichier représente une fenêtre du jeux.

5.1 La charte graphique

Les couleurs choisies sont principalement le bleu et l'or, le mariage de ces deux couleurs rappellent la royauté afin de montrer que les combats menés par le joueur ne sont pas de simples batailles de campagne mais bel et bien des guerres menées par de puissants seigneurs. Ce choix de couleurs permet aussi un habillage sobre et efficace, ainsi on évite de surcharger l'interface et de perdre le joueur avec des couleurs dans tous les sens sans liens entre elles. Le choix d'une interface simpliste est fait pour mettre en évidence les éléments principaux de la fenêtre, la couleur "gris clair" permet notamment de créer un contraste entre le voyant de l'or qui est renforcé par le sombre du bleu à l'intérieur. La police choisie est "War of Asgard", c'est une police de style médiévale qui reste cependant très lisible de par le fait que les caractères ne sont pas tassés et que le dessin est suffisamment épuré pour ne pas ressembler à un amas de lettres. Le style médiévale de cette police a été choisi pour correspondre au thème de notre jeu qui est l'héroïque fantaisie. Les icônes représentant les unités sont pour le moment à l'état d'ébauche, pour ces dernières nous avons demandé l'aide de Morgane Jolivel, une jeune dessinatrice, afin d'obtenir un rendu à la fois cohérent (Plutôt que des images piochées par ci par là avec des styles graphique différents) et répondant à nos demandes, donc une images spécifique à chacune des unités (Et donc éviter de prendre quelque chose "à peu près ressemblant"). Cette association nous permet ainsi, en plus du travail de groupe, d'apporter un côté "Travail avec un/e groupe/personne tierce". Pour les musiques, nous utiliserons des musiques libre de droits ou bien comme pour les icônes nous les créeront nous même afin d'éviter tout problème de droits.

5.2 Préparation du combat

Sur cet écran se déroule le choix des joueurs, il y a au total deux choix principaux. Le premier choix est le joueur humain, dans ce cas des questions lui seront posées afin de savoir ses actions dans le jeu. Le second choix est celui de l'intelligence artificielle, ce choix se découpe sous trois catégories : simple, médium, difficile. Une fois que les deux joueurs sont choisis, on passe au choix de la race de son armée. Cet écran apparait seulement si un joueur humain est sélectionné, l'Intelligence Artificielle choisie elle-même sa race ainsi que la composition de son armée.

Dans ces deux fenêtres, l'utilisateur aura simplement besoin de cliquer sur l'une des boites pour valider son choix. La simplicité de ces fenêtres est voulue car ainsi le joueur n'est pas perdu parmi un choix trop vaste. Ainsi les choix de ces paramètres se font de façon fluide.



5.2.1 Explication du code

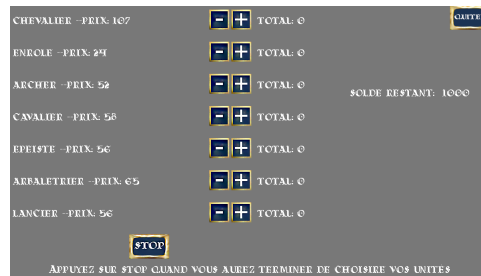
Ces deux fenêtres font parties des plus simples au niveau du code, elles ont toutes les deux une structure générale plus ou moins identique. En premier lieu on crée une variable vide qui contiendra les éléments à renvoyer pour la suite du programme. Ensuite on charge les images qui serviront de boutons pour l'utilisateur, on crée les événements liés au clic de la souris sur les images que l'on a chargé précédemment, puis pour finir on réalise un test pour savoir si toutes les conditions sont réunies pour passer à l'écran suivant et on retourne les valeurs de la variable créée au départ dans le programme principal pour la traiter et passer à la suite du programme.

```
def fenêtre(écran : "écran pour l'afficher") :
    variable = valeur à récupérer
    img = créerImage(chemin,nom) positionner
    image
    si clique sur image : mettre nom image dans variable
    si condition réunis : retourner variable
```

5.3 Phase de création de son armée

Une fois que l'utilisateur a choisi sa race, on passe sur l'écran de création des armées. Cet écran se divise en trois colonnes, dans la première colonne est affiché le nom des unités que le joueur peut acheter avec ses points ainsi que leur prix. La deuxième colonne est composée de trois éléments. Deux boutons, un bouton plus et un bouton moins, ils permettent d'ajouter et d'enlever des unités à la composition de l'armée, le dernier élément représente combien de fois cette unité est présente dans l'armée du joueur. La dernière partie de la fenêtre montre le solde restant du joueur, ainsi il pourra optimiser au mieux la composition de son armée par rapport à son solde. Une fois que le joueur aura terminé de choisir la composition de son armée, il lui suffira de cliquer sur le bouton stop.

Le code de cette fenêtre est très ressemblant à celui des fenêtres précédentes à la différence que l'on ajoute une instruction pour compter les unités et le solde restant.



5.4 Phase de jeux :

Comme indiqué dans les règles le jeux se passe en tour par tour avec une phase de déplacement et une phase d'attaque. Le jeux se fera sur cette fenêtre.

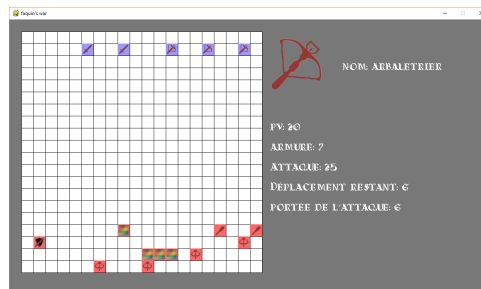


FIGURE 2 – écran de jeu principal

Cette fenêtre est scindée en deux partie, la première avec le plateau de jeu, c'est sur celle-ci que le joueur cliquera pour sélectionner l'unité avec laquelle il fera son action. La seconde partie est là pour donner les informations sur l'unité qui a été sélectionnée, on retrouve sur cette partie les statistiques principales de l'unité, ainsi que son nom et une image la représentant (Identique à l'icône).

5.4.1 Explication du code

Pour cette partie, on divise le code en 5 fonctions distinctes. La première sert à créer la grille et afficher les pions sur cette dernière, pour cela on demande en entrée la liste des unités, ensuite on parcourt cette dernière en regardant les coordonnées de chaque unités puis on la place dans la grille réservée au placement des unités. En même temps, on converti les coordonnées de l'unité en pixel pour placer le pion sur la fenêtre après avoir préalablement dessiné la grille.

La seconde fonction est celle qui permet de placer les pions au début de la partie. Pour cela on donne la liste des unités du joueur qui doit placer ses pions. Puis on demande au joueur de cliquer sur la grille à l'endroit où il désire placer son pion. Bien évidemment on fait des tests sur l'endroit cliqué par le

joueur pour savoir si : Il clique bien sur la grille et si il est bien dans sa zone de placement. A chaque fois que le joueur clique sur une case autorisée, on fait appel à la fonction qui affiche les pions sur la grille cependant si le joueur clique sur une case non autorisée, un message d'erreur apparait sur l'écran lui en informant.

La troisième fonction sert à créer l'affichage sur la partie droite de l'écran. Pour cela il nous faut deux paramètres, la race et le nom de l'unité. Ainsi on récupère précisément l'unité désirée dans un dictionnaire contenant les unités de chacune des deux armées, on récupère les statistiques désirées de cette unité puis on les affiche sur la partie droite de l'écran.

La quatrième fonction est la fonction de déplacement. Cette fonction permet au joueur de déplacer ses pions durant la phase de déplacement, celle-ci diffère de la phase de placement par le fait que le joueur est libre d'aller où il veut sur la grille du moment que sa destination finale n'est pas une case où une unité ennemie est déjà présente ou un obstacle. Durant cette phase le joueur peut choisir quelle unité il veut bouger en premier, il lui suffit de cliquer dessus. Puis il lui suffit de cliquer sur la case où il veut aller, bien évidemment des tests sont effectués pour valider ou non son choix.

La cinquième fonction est la fonction permettant d'attaquer une unité ennemie, pour le moment cette fonction est encore à l'état d'ébauche. Cette fonction n'étant pas implémentée, sa description sera très sommaire. Pour cette fonction on donne les unités présentes sur le plateau puis comme pour la phase de déplacement, on clique sur notre unité attaquante puis sur l'unité qui sera attaquée.

6 Tournois

La fonction `tournois` prend en paramètre une population d'armées et possède 7 arguments par défaut (quelle IA, quel module utiliser pour les règles, quel plateau utiliser, combien de fois exécuter chaque combat, sauvegarde ou non de données ainsi que le fichier où sauvegarder ces données).

Le tournoi permet de faire s'affronter plusieurs armées et de les classer par ordre du nombre de victoires. Il est principalement composé de deux boucles for : une première pour parcourir la liste des armées donnée en paramètre, la deuxième pour effectuer plusieurs fois un combat. Au début de chaque combat, si on a donné en paramètre à la fonction `tournois` `save = True`, la composition des deux armées, ainsi que la `seed` du combat, est écrite dans une variable dont le contenu sera sauvegardé dans un fichier précisé en paramètre de la fonction. En fin de combat la composition finale des armées est aussi sauvegardée à la même fin.

Une fois que le tournoi entre toutes les armées est terminé on écrit dans le fichier de sauvegarde le nombre total de victoire de chaque armée.

Une fois toutes les informations enregistrées dans le fichier les populations sont rangées par ordre de nombre de victoires et la fonction renvoie la liste des armées classée ainsi que les informations qui ont été sauvegardées si `backData=True`.

7 Algorithme génétique

L'algorithme génétique est un objet qui permet d'effectuer un certain nombre de combats et de sélectionner les armées les plus optimisées pour le jeu. Pour cela on donne à l'algorithme le nombre de tours que l'algorithme devra effectuer, le nombre d'armées qui seront utilisées au départ de l'algorithme, le pourcentage d'armées qui resteront inchangées, le pourcentage d'armées qui seront mutées, le pourcentage d'armées qui seront croisées, et un paramètre qui indique si des armées 'pures' seront utilisées durant le premier tour de l'algorithme. On entend par armées 'pures' des armées constituées d'un seul type d'unités.

L'algorithme commence donc par générer le nombre requis d'armées : suivant les paramètres donnés soit les armées sont générées de manière totalement aléatoires, soit on y ajoute les armées 'pures'.

Ensuite l'algorithme utilise la fonction `tournois` pour faire s'affronter les armées et déterminer lesquels sont les meilleures. Une fois le classement effectué l'algorithme met dans une nouvelle liste d'armées les meilleures armées du tournoi qui restent inchangées. L'algorithme procède ensuite à des mutations sur certaines armées et produit de nouvelles armées en effectuant des croisements entre d'autres armées. Une fois tout ceci effectué l'algorithme boucle jusqu'à ce qu'il ait effectué le nombre d'itération requis. Après cela l'algorithme sauvegarde les informations concernant les mutations et les croisements qu'il a effectués et affiche le temps qu'il a mis à s'exécuter.

8 Croisements

Dans ce projet, les croisements entre armées se basent sur le même principe que les croisements génétiques. Plus précisément, deux armées parentes doivent être fournies (ceci en partie géré par l'algorithme génétique), à partir de ces armées, une armée enfant sera créée. Le croisement est réalisé uniquement sur la composition des armées (donc, par exemple, pas de croisements au niveau des soldes, ces derniers étant fixe d'une armée à l'autre). Ces compositions, ensemble d'unités, vont devoir se "mélanger" de façon aléatoire tout en respectant une contrainte qu'est le solde disponible d'une armée. A cause de cette contrainte, on ne peut réellement tirer au sort une unité d'armée parente pour rejoindre l'armée enfant sous peine de voir un solde restant négatif (théoriquement impossible). Pour palier à cette contrainte, nous détectons dans un premier temps les unités dites "embauchables" d'une première armée parente en fonction du solde restant de l'armée enfant. Une fois cela réalisé, on tire aléatoirement l'une de ces unités embauchable que l'on ajoute à l'armée enfant (le solde étant ajusté en conséquence). On recommence cette fois-ci avec la seconde armée parente, on détecte donc les unités embauchables pour l'armée enfant, unités que l'on va tirer au sort pour rejoindre l'armée enfant etc. S'il ne reste plus d'unités embauchables dans l'une des armées parente, ce cycle va tout de même continuer si l'autre possède encore de ces unités embauchables et ce jusqu'à ce que les deux parents n'aient plus d'unités ou du moins d'unités embauchables.

9 Mutations

Tout comme pour les croisements, les mutations, se basant sur les mutations génétiques, ne vont affecter que uniquement les compositions d'armées. Selon un taux de mutation donné, et en fonction de la taille de composition d'armée, un certain nombre d'unités disparaîtront pour laisser place à de nouvelles unités. Encore une fois, la contrainte du solde est présent, pour palier à ce problème lors de la mutation, pour N unités supprimées (et restitution de leur valeur dans le solde restant), nous tentons d'ajouter N fois une unité embauchable. Dans le cas où à la $(N-X)$ ième, X différent de 0, tentative d'ajout d'unité il ne resterait plus d'unités embauchable, l'armée sera alors restituée avec les mutations réalisées et par conséquent avec X unités en moins dans sa composition qu'avant cette série de mutations.

10 La sortie web

10.1 Introduction

Nous avons réalisés une sortie afin d'exploiter au mieux les données qui ont été créées par l'algorithme génétique. Le premier objectif de cette sortie web est surtout de pouvoir visualiser les données après leurs sortie, par la suite le but de cette sortie est de présenter une évolution des armées et ainsi remarquer le comportement de l'algorithme génétique.

10.2 Disposition de la page web :

10.2.1 Affichage de données générale

Cette page web sépare les combats par tournois afin de libérer de l'espace sur la page web est facilité la lecture des données.

Données après tournois

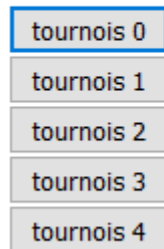


FIGURE 3 – la page au premier chargement

Sur cette page l'utilisateur pourra cliquer sur le bouton afin d'afficher les information du tournois concerner, les données affichées sont les compositions initiales de chaque armées ainsi que leur score. Le score représente le nombre totale de victoire que l'armée à obtenue lors de ce tournois.

Ainsi via cette page web on peut remarquer si une certaine composition est meilleur qu'une autre. Pour cela il suffit de regarder la partie "Compos initiale". Si une armé ce répète plusieurs fois on peut alors juger qu'elle est optimisé.

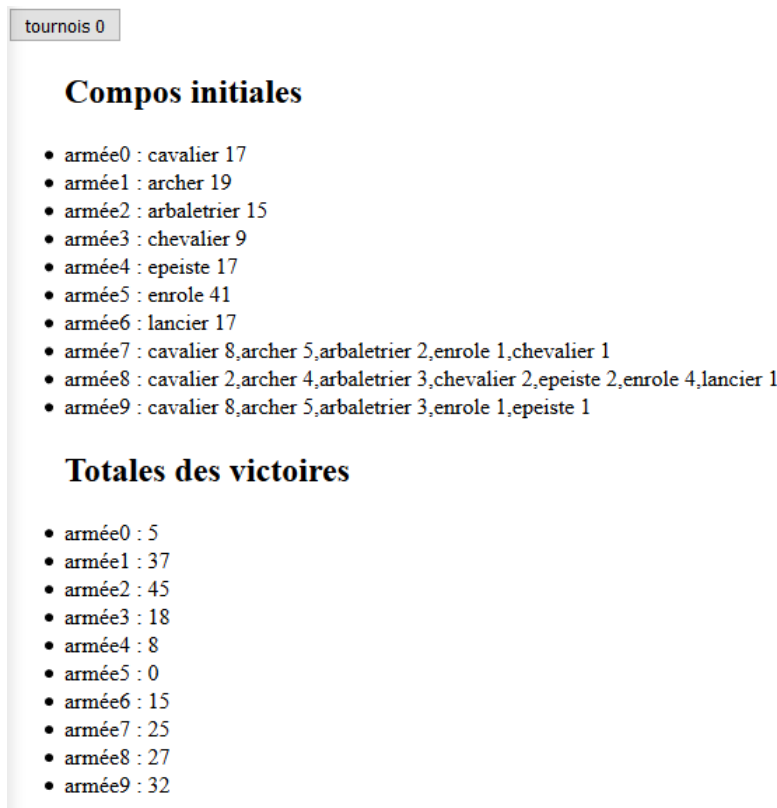


FIGURE 4 – la page après le clique sur un bouton tournois

10.3 résultats de l'exploitation des données

L'utilisation de cette page web nous a révélé des informations très importantes concernant notre jeu. Après avoir réalisé plusieurs tournois avec différents paramètres comme celui d'intégrer des armées pures (qui ne possèdent qu'un seul type d'unité).

Toutes les résultats des expériences dont nous allons parler ont été obtenus en utilisant l'intelligence artificielle simple (mouvement aléatoire des unités) et des unités de la race des humains.

10.3.1 Utilisation d'armées aléatoires uniquement

En utilisant uniquement des armées générées aléatoirement par le programme, on remarque que si l'algorithme génétique tourne peut de fois (peut croiser et du mutation), l'évolution vers une armée optimisée ne se fait que peut ressentir. Les armées ont tendance à tendre vers des armées qui se retrouvent avec un grand nombre d'unités à faibles coûts et les armées qui possèdent ma-

ritairement des unité à distance reste la plupart du temps inchangé ou peut. Cependant si l'ont augmente le nombre d'utilisation de l'algorithme génétique, on remarque alors une montée en puissance des armées possédant des unité de distance ainsi que le fait que quasiment l'intégralité des armées deviennent des armées majoritairement composés d'unité à distance. Le second point notable de cette expérience sur un grand nombre de tour et la répartition des victoires, plus le nombre de tours augmente plus on remarque que le nombre de victoire et réparties entre toutes les armées ainsi on peut remarquer que le jeu finit par s'équilibrer est qu'il n'y a pas une composition majeure mais un choix d'unité qui est préférable.

10.3.2 Ajout des armées pur

L'ajout des armées des armées pur au milieu d'un grand nombre d'armées aléatoire ou d'un faible nombre d'armées, ne change en réalité que peu de chose car les résultats se font ressentir très rapidement, le phénomène d'armées composé d'unité à faible coup ne se fait pas ressentir ici. Dès le troisième ou quatrième tour de l'algorithme génétique, on remarque déjà la suprématie totale de l'armée pur d'arbalétrier, cette armée et tellement dominante qu'elle restera l'une des meilleures même après de nombreux tours.

Cependant on remarquera que cette armée malgré des taux de victoire proche de 100%, elle sera dominée par une armée plus optimisée qui en gardant de très près le même nombre d'arbalétrier, sera complétée avec des archers qui coûtent moins cher propose une armée plus nombreuse tout en restant avec une armée composée exclusivement d'unité à distance.

Ainsi l'ajout de ces unités pure nous a confirmé la suprématie des unités à distance lors de l'utilisation de l'intelligence artificielle simple est d'armée de la race humaine et ceci peut importe le nombre d'armées aléatoire rajouté en supplément ou le nombre de tours effectués par l'algorithme génétique.

10.3.3 Conclusion

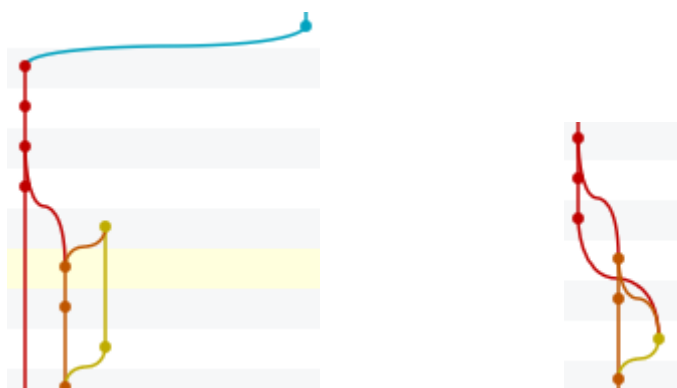
Pour conclure nous pouvons donc dire que avec les ressources en notre possession et les différents tests réalisés que l'armée la plus optimisée pour vaincre l'intelligence artificielle simple sur terrain plat avec des humains est : 9 arbalétriers et 7 archers.

11 Conclusion

Ce projet nous à permis de mieux appréhender la cohésion d'un groupe et nous à donné l'occasion de travailler sur un projet relativement imposant, ce que nous ne connaissions pas encore. Il nous à permis d'appliquer les connaissances obtenue durant l'année.

12 Les problèmes rencontrés

N'ayant pas l'expérience de projets aussi important, nous avons eu du mal à gérer tous les fichiers. En effet nous avons eu pendant longtemps plusieurs versions des fichiers ainsi que plusieurs fichiers obsolète. De plus notre cohésion d'équipe n'était plus aussi bonne que lors du premier semestre nous avons donc 2 séries différentes de push sur la forge. De plus, le code est pleinement optimisable. En effet une partie du projet n'est pas codé en objet alors que nous utilisons une langage objet. Nous avons aussi rencontré quelques problèmes avec git.



Exemple de problèmes avec git

13 Annexe

13.1 Vocabulaire

- PV : Points de vie.
- IA : Intelligence Artificielle.
- Tactical RPG : Jeu de rôle tactique.

13.2 Wiki

- Le Wiki du dépôt a été mis en place de sorte à faciliter l'accès au(x) :
- Informations concernant les packages utilisés
 - Versions antérieures du projet
 - Version en cours de développement
 - Informations annexes (Concernant la visualisation de statistiques via le web par exemple)

13.3 Les ressources d'informations

- Wikipédia : <https://fr.wikipedia.org/wiki/Wikip>
- Openclassroom : <https://openclassrooms.com>
- Pygame : <http://www.pygame.org/docs>
- youtube : <https://www.youtube.com>

13.4 Les outils utilisés

- Les outils que nous avons utilisé sont :
- Git : C'est un logiciel de gestion. Il permet de gérer les différentes versions d'un fichier, programme ou autre.
 - Python : Python est un langage de programmation objet.
 - PyGame : Une des bibliothèques de Python. Elle permet entre autre de créer une interface graphique.