FOUQUET Stevy
GRUDE Victorien
PAGIS Etienne

Deuxième année de licence SPI

Rapport de projet : jeu Terra-Battle

Sommaire :

Introduction :

Dans le cadre de notre cursus universitaire : la deuxième année de licence SPI , nous avons eu a réaliser un projet se présentant sous la forme d'un jeu . Parmi une liste de différents jeux , nous avons donc choisi de travailler sur le Terra-Battle . Ce jeu se place dans une matrice dans laquelle nous possédons différentes unités alliées et dont le principe et de battre toutes les unités adverses se situant sur le terrain . Nous entretenons au fur et a mesure que le jeu avance, une escouade où se situe toutes nos unités, c'est grâce à cela que nous pouvons décider qui l'on garde, ceux qu'on ne veux pas mettre au combat, gérer leur équipement etc.

Nous avions eu comme consigne de reproduire le jeu dans une version modifié : nous avons eu le droit de modifier certaines règles du jeu, certain fonctionnement et de traiter différemment certains points lié à l'Intelligence Artificielle (les déplacements etc)

Afin de mettre en œuvre ce projet nous nous sommes donc poser différentes questions :

-Quels aspects du jeu allons nous être capable de réaliser et quel modification des règles/fonctionnalités allons nous devoir apporté ?

-Comment réaliser les structures lié aux différentes unités dans le jeu ?

-Comment gérer tout les différents déplacements, changements qui vont être produit ?

-Comment concevoir une intelligence artificielle convenable pour le jeu ?

Dans la suite de ce rapport il sera expliquer les différents points suivants :

- Réponses aux différents problèmes et mesure pris pour la programmation
- Organisation des taches et du travail
- Explication des différentes partie du code
- Résultat puis conclusion

Mise en avant du problème et début de réponses :

1.Ce qui est lié au joueur :

Tout premièrement prenons le problème lié à l'affichage et les décisions prise en fonction de la matrice du jeu

Afin de pouvoir ressembler au jeu au maximum, nous avons décider de prendre une taille de matrice quasi équivalente de taille 9*7 alors que l'originale est d'une taille de 8*6 . Ce choix a été pris afin de permettre une plus grande lisibilité pour le joueur : puisque la fenêtre de terminal est un peu sombre et grande comparé à un smartphone (plate forme du vrai jeu), nous prenons la première ligne et la première colonne afin de marquer les coordonnées notée au préalable dans la fonction d'affichage de matrice . Cette fonction sert aussi à placer visuellement les entité et à les placer au lancement du jeu .

Parlons ensuite de la partie du déplacement des entité alliée dans la matrice de jeu

Dans le vrai jeu , le joueur a quelques secondes afin de pouvoir déplacer tous ces personnages disponibles sur le terrain en faisant glisser son doigt sur la direction que doit prendre son entité . Nous nous sommes vite rendu compte que ceci n’était pas réalisable pour le terminal puisque il faut taper ses commandes de déplacements et qu'en quelques secondes il ne serait pas possible de déplacer autant d'unité qu'on le souhaite . Pour remédier au problème nous avons programmé le déplacement des entités de manière à avoir un nombre de déplacement limité : nous avons trouvé judicieux de mettre 15 déplacements par entité dans toutes les directions afin de pouvoir permettre tout de même à une entité de parcourir une grande distance dans la matrice . Pour choisir l'unité à déplacer il suffira de rentrer ses coordonnées de départ.

2. Ce qui est lié à l'IA

Pour répondre aux questions lié aux déplacements que doivent effectuer les entités ennemies qui sont gérées par l'ordinateur nous avons eu de nombreuses questions à résoudre

Tout d'abord la stratégie que doit adopter l'IA dans le jeu afin de pouvoir rivaliser avec le joueur Nous avons convenu que pour l'IA ces déplacements doivent permettre d'attaquer le plus d'entités adverse sans non plus ce mettre trop en avant afin de ne pas trop subir des attaques du joueur au tour suivant . Pour ce faire il fallait mettre en avant le terme de « cible » : à quel mesure une entité a éliminer est ou n'est plus une cible au yeux de l'adversaire , pour nous la réponse est que la cible doit être prise en tenaille (de la forme chasseur-cible-chasseur) mais qu'elle est quand même la possibilité de fuir afin de permettre au joueur de réfléchir quand même à une autre stratégie d'attaque. Ce qui nous semble être judicieux au niveau du déplacement et de ne pas déplacer tous les chasseurs si toutes les cibles sont déjà prise en tenaille pour mettre un peu plus de difficulté dans le jeu puisque les adversaires font en sorte de ne pas se mettre en danger et d'aussi facilité les décisions du joueur comme expliquer précédemment

Vient ensuite la question du repérage de « cibles » pour l'IA , quand es-ce qu'elle comprend qu'une entité est une cible et quand elle ne l'est plus ? Pour ce faire , nous mettons au point une recherche de toutes les unités dans la matrice et qui permet de faire le tri des unités cibles en fonction de leur environnement

3. Ce qui est lié aux attaques

Afin de se simplifier la tâche au niveau des attaques et étant donné que les déplacements effectuer par les entités de type ia font en sorte de prendre la « cible » en tenaille, nous avons définit les attaques de l'ia comme celle du joueur c'est à dire que la cible qui est censé se faire attaquer par la partie attaquante doit être prise en tenaille par les attaquants, l'attaque se déroule en fonction de l'orientation de l'attaque (si elle est faite de façon horizontale ou verticale dépendant du sens de la tenaille) . Si sur cette orientation il y a plusieurs alliés, ils aident aussi pendant l'attaque en donnant des points de dommage en plus .

Donc pour ceux qui est de l'attaque, que se soit l'attaque d'une unité allié ou d'une unité ennemies, le premier problème était de savoir la situation dans laquelle se trouve la cible .

Pour cela nous regardons pour chaque cible d'attaque si autour d'elle se situe des entités attaquantes et si oui dans quelle sens est situé la tenaille pour savoir dans quelle direction il faudra chercher le nombre de voisins qui pourrait attaquer .

4Les structures

4.1 : Les personnages

Les personnages sont des entités qui possèdent plusieurs caractéristiques qu'il a fallu retranscrire sous forme de structures et d'énumération. Les caractéristiques retranscrites sont les suivantes : le nom, la race, le job (spécification de l'entité), l'arme, l'attribut, le rang, les statistiques, le niveau et l'expérience.

Le nom représente un moyen d'identifier les personnages donc il est essentiel aux personnages.

La race n'est pas si crucial mais le concept de race étant présent dans le jeu original, nous avons choisi de l'ajouter à notre projet pour définir les personnages des ennemis, les personnages et les ennemis ont donc des races spécifiques.

Le job est une caractéristique très importante car elle permet de définir souvent les statistiques du personnage et définit son type d'arme, notre projet compte 13 jobs, dont 5 basiques et 8 avancés, les jobs basiques sont les classes des personnages et des ennemis, arrivé dans un certain niveau, les jobs avancées sont les évolutions des jobs basiques ayant des caractéristiques différentes

L'arme sert à définir les attaques du personnage avec des bonus de dégâts, si l'adversaire possède une arme contre laquelle le personnage est avantagé, le personnage fera plus de dégât . L'attribut est complémentaire à l'arme.

Le rang détermine la force d'un personnage, un rang B aura de plus faibles statistiques qu'un rang A par exemple. Il y 4 rang que nous avons définit pour les personnages (par ordre croissant : B,A,S,SS)

Les statistiques sont les caractéristiques les plus importantes car elles déterminent les points de vie, l'attaque physique, l'attaque magique, la défense physique et la défense magique cela spécifiant la capacité au combat des différents personnages. Ces caractéristiques augmentent avec les niveaux des personnages. Les statistiques sont directement influencées par le job et par le rang : un guerrier de rang A et un guerrier de rang B ne progresse pas à la même vitesse.

Le niveau sert à déterminer la puissance du personnage car, plus un personnage à un niveau élevé, plus ses statistiques sont élevées, donc plus le personnage est puissant.

Enfin l'expérience sert de palier afin de définir si un personnage peut monter de niveau ou non. Plus le rang d'un personnage est élevé, plus la quantité d'expérience a obtenir est grande, elle augmente également lorsque le niveau augmente.

La race, la classe, l'arme, l'attribut et le rang sont des énumérations, alors que les statistiques (composé de floats) ainsi que le personnage en lui même sont des structures. Le nom est une chaîne de caractère et le niveau ainsi que l'expérience sont des entiers.

Les personnages sont générés aléatoirement si le joueur souhaite en recruter un nouveau.

Mais les personnages seuls ne sont pas utiles dans Terra Battle, il en va de même dans notre projet. Les personnages doivent formés des escouades afin de combattre.

4.2 Les escouades

Les escouades permettent de former des équipes de personnages avec un minimum de deux personnages et un maximum de six personnages.

Comme il faut composer les équipes, il faut permettre au joueur de pouvoir composer la sienne avec les différents personnages qu'il possède. Il faut également pouvoir sauvegarde la liste de ses personnages afin qu'il ne les perde pas en fermant le terminal.

Organisation du travail :

Les tâches étant toutes différentes les unes des autres, afin de nous organiser nous avons diviser ses tâches en parties : une partie relatant toutes les structures de leur appels et de leur modification , une partie étant sur les déplacements des deux opposants (joueur et IA) et une dernière partie relatant de l'attaque et du calcul de voisin dans la matrice

La partie de toutes les structures dont le nom de fichier se nommant classes.c a été géré par Stevy Fouquet

Les autres parties ont été géré de la façon suivante :

-Affichage de la matrice : Etienne Pagis ,Victorien Grudé

-Affichage du menu : Etienne Pagis

-Calcul des voisins et fonction d'attaque : Victorien Grudé

-Déplacement de l'IA : Etienne Pagis

-Déplacement du joueur : Victorien Grudé

Explication du code :

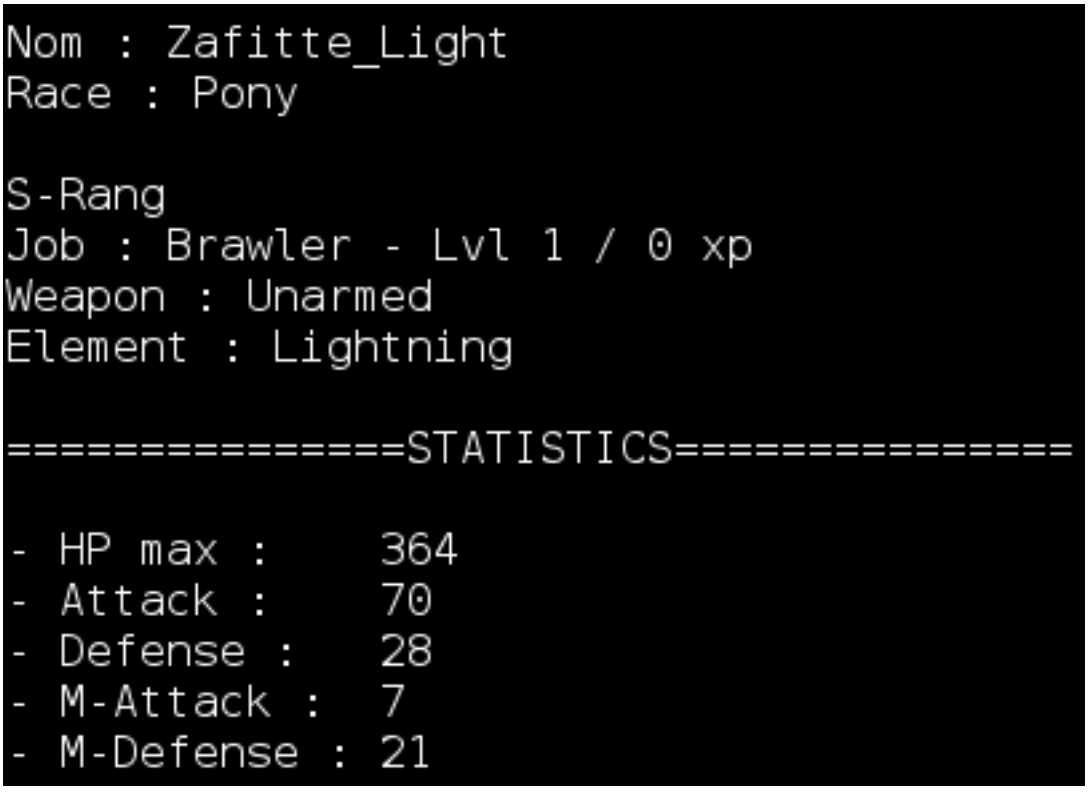
1 : Les structures

1.1. Codage Personnages

La mise en forme des différentes structures n'a pas été le plus difficile, cinq énumérations et quatre structures ont été nécessaires afin de stocker toutes les informations pour gérer les personnages.

Le principal problème rencontré lors du codage de la gestion des personnages était l'allocation de la mémoire par rapport au « nom » des personnages, lorsqu'il fallait les créer, les charger depuis un fichier, il y a aussi eu quelques problèmes par rapport aux statistiques mais ce problème fut réglé avec des « define ».

Parmi les fonctions par rapport aux personnages, il y en a des simples : une fonction permettant de créer tout simplement le personnage avec toutes ses caractéristiques en paramètres, une fonction affichant ces différentes caractéristiques et une fonction permettant de générer aléatoirement les personnage mais avec des degrés différents par exemple, un personnage de rang « S » sera plus difficile à obtenir qu'un personnage de rang « B » car un personnage de rang « S » est plus puissant.



La montée de niveau est géré par deux fonctions : lorsqu'un personnage monte de niveau, il gagne en force et donc en statistiques, donc après avoir vérifier avec la fonction montee_level si le palier d'expérience d'un personnage lui permettait de passer au niveau suivant, le programme fait appel à la fonction montee_statistiques si le palier est dépassé. montee_statistiques vérifie à quel job a l'entité et quel est son rang, ensuite elle fait le calcul de ratio en fonction de ces deux variables.

Il y a également une fonction changement_classe qui permet aux personnages avec des jobs basiques d'évoluer en job avancé si le personnage atteint un certain niveau. Pour cela nous demandons au joueur si il veut faire évoluer son personnage, si il accepte de le faire évoluer, il se peut qu'il doive choisir entre deux ou trois jobs celon son job basique.

Les statistiques augmentent lorsque le personnage change de job, pour cela la fonction montee_level appelle la fonction changement_classe pour signaler que le personnage change de job. Cette fonction retourne donc le même personnage mais avec le nouveau job et au niveau 1 avec la même expérience qu'avant le changement, la fonction montee_level fait donc appel à elle-même afin que le personnage soit à nouveau, au niveau qu'il était avant le changement.

Les différents jobs n'offrent pas les même changements par exemple il faut être Archer pour devenir Sniper.

Les fonctions faisant appels aux fonctions ci-dessus sont également utilisées dans les fonctions se référant aux escouades.

1.2. Codage des escouades

Pour les escouades, les fonctions tel que afficher_escouade et montee_level_escouade sont des adaptations de certaines fonctions, en rapport avec les personnages, qui sont appelées dans celle-ci.

Afin que le joueur puisse gérer son escouade, deux fonctions sont cruciales : pouvoir ajouter des membres à son escouade, et pouvoir en retirer.

La fonction ajouter_membre demande au joueur de saisir le nom d'un personnage présent dans sa liste de personnage puis l'ajoute à son escouade, le joueur ne peut pas avoir plus de six personnages dans son escouade et ne peut pas ajouter un personnage déjà présent dans son escouade.

La fonction retirer_membre quant à elle demande au joueur de saisir le nom d'un personnage présent dans son escouade afin de le retirer, il pourra ensuite si il le souhaite réintégrer ce personnage a son escouade avec la fonction ajouter_membre. Le joueur ne peut pas avoir moins de deux personnages dans son escouade.

La fonction recherche_membre_escouade permet de rechercher des personnage dans l'escouade comme son nom l'indique. Elle retourne le numero du personnage de l'escouade qui correspond au nom entré dans la fonction retirer_membre, et retourne un numero spécifique si elle ne trouve pas le personnage. Quant à la fonction recherche_membre_liste, elle possède la même fonctionnalité que recherche_membre_escouade mais elle parcourt la liste au lieu de l'escouade puis retourne le personnage souhaité.

La fonction xp permet de répartir équitablement les points d'expérience acquis à la fin d'un combat à tout les membres de l'escouade.

1.3 Codage Liste

La liste est un élément important car elle comptient tout les personnages du joueur, cette liste peut être sauvegarder grâce à la fonction sauvegarde_liste qui écrit dans un fichier sauvegarde.txt les différentes caractéristiques de tout les personnage du joueur. Ce fichier sauvegarde.txt est utilisée dans la fonction charger_liste afin de récupérer tout les personnages du joueur qu'il a sauvegardé au préalable. Mais la fonction charger_liste est accompagnée d'une autre fonction : la fonction aide_chargement. Cette fonction permet de régler un problème de la fonction charger_liste, en effet la fonction charger_liste ajoute un personnage en trop dans la liste du joueur, elle répète le dernier personnage de la liste présente dans le fichier sauvegarde.txt. La fonction aide_chargement est donc là pour palier à ce problème en comparant les noms de tout les personnages entre eux, puis si un nom est présent deux fois, diminue le nombre de personnage présent enlevant ainsi le personnage en trop.

Le joueur peut recruter des personnages, pour cela le joueur doit valider son choix de recrutement avec la fonction choix_recruter, ensuite si le nombre de personnage dans la liste du joueur est inférieur à la limite de la liste, la fonction generer est appelée. Le personnage généré doit être « testé » car si le joueur recrute un personnage qu'il possède déjà, le personnage ne doit pas apparaître deux fois mais doit monter au niveau supérieur. Pour cela, on appelle la fonction booléenne est_present qui renvoie si le nouveau personnage est présent dans la liste, la fonction se base sur le nom du personnage, son attribut, et son job mais après les quelques modifications de notre programme, le nom seul est nécessaire.

La fonction renvoie 0 si elle ne trouve pas le personnage dans la liste et il est donc ajouté à la liste des personnage du joueur. Sinon, le personnage est déjà présent dans la liste et doit donc monter de niveau. Pour le monter de niveau, la liste passe dans la fonction suppression_doublons, elle est similaire à la fonction est_present mais renvoie la liste au lieu de renvoyer 0 ou 1. Donc la fonction suppression_doublons fait appel à la fonction verification.

La fonction vérification reçoit en paramètres la liste, le numéro du personnage, le nom du personnage, ainsi que sa race car si les races sont différentes et bien nous les rendons identiques afin que le cas dans lequel deux personnages possédant les mêmes noms mais ayant des races différentes ne se produise pas. La fonction distribue l'expérience nécessaire pour monter d'un seul niveau, fait appel à la fonction montee_level puis renvoie la liste de personnage.

Après avoir été recruté, les caractéristiques du personnage en question sont affichées.

La fonction afficher_liste permet d'afficher la liste des personnages de manière brève, cette affichage contient, un numéro pour la lisibilité, le nom du personnage, son niveau, son job son arme et son attribut.

La fonction mise_a_jour_liste sert à mettre à jour, les personnages de la liste qui sont présent dans l'escouade car les personnages dans l'escouade du joueur combattent et gagnent donc en puissance. Il faut donc mettre à jour leurs nouvelles caractéristiques dans la liste des personnages.

La fonction init_liste est une fonction donnant au joueur, une liste de personnage de départ.

Toutes ses fonctions sont utilisées dans une fonction menu_personnage lequel propose plusieurs choix permettant au joueur de gérer son escouade, de sauvegarder sa liste de personnages, de charger sa liste de personnages ou de

1.4. Codage Ennemi

Les ennemis ne possèdent que peu de fonctions en comparaison avec les personnages, même si elles sont basées sur celles-ci.

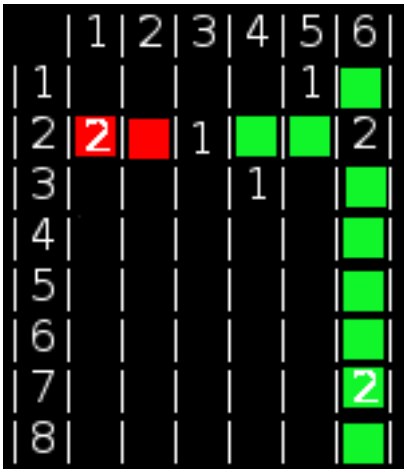
Les ennemis sont générés aléatoirement en fonction du niveau de difficulté choisi par le joueur. Plus la difficulté est élevée, plus les ennemis seront de haut niveau. Mais ces ennemis sont rassemblés dans une escouade adverse nommée horde. Les différents ennemis générés sont placés dans une horde grâce à la fonction generer_horde. Le joueur peut choisir la difficulté dans le menu principal.

2. Calcul des voisins et attaque

Afin de pouvoir trouver le nombre de voisin nous avons fais une fonction qui se nomme voisin qui prend en paramètres les variables i et j, qui sont les coordonnées de l'entité dont on veut calculer les voisins. Pour ce faire nous nous plaçons au coordonnées prises puis nous calculons la ligne entière et la colonne entière où elle se situe. En retourne la valeur du nombre de voisins si il y en a et renvoi 0 si il n'y en a pas . Si au moment de l'appel de la fonction , celle ci renverra la valeur -1 .

Cette fonction tenaille fait appelle à plein de fonction de type est_tenaille : il y en a une pour chaque situation possible dans le jeu. Elles fonctionnent toutes de la même manière, elles prennent en variables les coordonnées de la personne qui est attaqué et place dans une pile toutes les unités qui sont situer en tenaille si il y en a plusieurs de suites . Elles renvoient la valeur 1 si il y a prise en tenaille peut importe le nombre de cible, elle renvoient 0 si il n'y a pas de tenaille .

Cependant ces fonctions ont besoin de repérer si l'unité est un allié ou un ennemi, donc pour ce faire, nous avons fait des fonctions qui permet de le définir : elles se nomment est_ennemis et est_allie. Elles prennent en paramètres les coordonnées de l'entité dont on vérifie l'appartenance a la troisième variable qui est le joueur (1 pour le joueur 2 pour l'ennemi). Elle renvoi si c'est un ennemi ou un allié, 0 si pas de réponse et -1 si c'est une mauvaise cible (dans notre cas des 0).



3. Les déplacements

Lorsque le joueur doit effectuer ses déplacements , on lance immédiatement la fonction void deplacement_joueur. On lance un affichage demandant au joueur si il veut ou non bouger son unité et quelle sont les coordonnées de celle ci. Ensuite on affiche toutes les possibilités de mouvement que le joueur peut effectuer c'est après cela que l'on vérifie si le déplacement est possible avec l'autre fonction dont on va parler

Les déplacements possible du joueur sont définit dans une fonction qui s'appelle void deplacement_possible qui prend en paramètres des pointeurs *i et *j qui sont les coordonnée de l'entité a déplacer, dep pour les déplacement que l'on souhaite exécuté *nb_dep le nombre de déplacements effectué. Elle vérifie pour les coordonnées qui ont était donné si le déplacement ne sort pas de la matrice, si il n'y a pas de collisions sur le chemin ou si l'entité n'est pas entouré d'ennemis. Si le déplacement n'est pas possible on affiche un message d'erreur puis on lui redemande de rentrer les coordonnées

4. Les attaques

À la fin de chaque tour (une fois à la fin du tour du joueur et une fois à la fin du tour de l'IA) on procède a l'attaque, soit du joueur soit de l'IA en fonction du tour qui viens de finir, pour l'attaque la fonction void attaque_allie ou void attaque_nemesis, en fonction de qui attaque, parcourt toute la matrice et envoie les coordonnées de chaque 1 ou 2 à la fonction void attaque_joueur ou void attaque_ennemis qui prennent en paramètres les coordonnées i et j de l'attaquant.

Les fonction void attaque_joueur et void attaque_ennemis cherche tout d'abords tout les ennemis adjacent à l'unité attaquante puis les empile dans une pile nommées pile_att. Une fois toute les cibles potentiels empiler le programme dépile les coordonnées des ennemis et vérifie pour chacun d'entre eux si il sont en tenaille avec la fonction est_tenaille. Si l'unité est un tenaille elle prend alors des dégâts calculer selon une équation complexe :

$$PV_{perdus} = \left(\frac{(Niv \times 0.4 + 2) \times Att \times 80}{Def \times 50} + 2 \right) \times voisin$$

Conclution :

Pour conclure, les points non traités sont la mise en place d'une interface graphique, a cause d'un manque de temps. De plus nous avons du abandonnées le codage d'un boss, de quatre cases, dans les niveaux car le codage des déplacement d'un éventuel boss de quarte case et son attaque étais trop compliqué.

Un problème du jeu est ça difficulté, nous n'avons pas vraiment cherché a le rendre équilibrer, il est difficile, en difficulté très facile, de gagner et pour gagner en une plus haute difficulté il faut tricher,c'est a dire changer son fichier de sauvegarde des personnage pour leur données de meilleur statistiques, sinon c'est presque impossible.

Si nous avions eu plus de temps nous aurions pu améliorer le programme en rajoutant une interface graphique. Nous aurions pu aussi codé le boss, le codage du boss pour le calcule des voisins et pour l'attaque étant déjà fais.