Candale Elliot
Ciron Fabien
König Linus

Compte rendu de Projet L2 SPI
Battle Tactics :

Sommaire :

I. L’appropriation du sujet « *Battle Tactics* »

a. Les Règles

Pour commencer, le Jeu « Battle Tactics » est un jeu qui se joue à deux joueurs, chaque joueur est en possession de 3 personnages en début de partie. Chaque personnage possède un point d'attaque, un de vie et un de range (correspondant respectivement aux dégâts infligés lors d'une attaque, aux dégâts que la pièce peux subir avant d’être éliminée et de la distance à laquelle il est capable d’attaquer une autre pièce).

Ensuite le joueur dois attribuer des points de compétence à chaque personnage afin d'améliorer ses caractéristiques.

Puis à tour de rôle les joueurs prennent le contrôle d'un de leurs personnages et peuvent se déplacer, changer l'orientation de la pièce ou commencer à s'entre attaquer.

Entre outre une spécificité de « Battle Arena » est de comptabiliser une attaque par derrière comme un coup critique, qui double alors sa puissance de base (si attaque vaut 2 les dégâts seront de 4).

Enfin chacune de ces actions coûtent des points d'actions, points à utiliser avec parcimonie car seulement 3 seront attribués à chaque tours.

Pour finir, la victoire s'obtient en éliminant tout les personnages de l’adversaire.

b. Définition des structures utilisées

Avant de se lancer dans le vif du sujet il est nécessaire de préciser que la structure principale de notre programme est t_perso ; elle est composée (comme énoncé plus haut) de variables correspondants a l'attaque, la vie, la range mais aussi d'un emplacement pour stocker le nom du dit personnage afin de permettre l'affichage de messages personnalisés et de les différencier, t_perso contient également les structures t_position et t_camp que nous verrons plus après. Pour finir, Il est nécessaire pour la bonne compréhension de préciser que l'arène est en fait une matrice de structure de type t_perso.

Passons aux énumérations, la première d'entre elles est t_camp, elle permet de différencier les personnages des deux équipes afin d'éviter, par exemple, le « friendly fire » mais elle permet également de repérer la position des pièces vivante sur la matrice en se composant des valeurs suivantes: joueur1, joueur2 et vide (correspondant à une case vide).

La seconde énumération de ce jeu est t_position, permettant de représenter l'orientation d'une pièce en prenant les valeurs : nord, sud, est ou ouest. Cette énumération est nécessaire pour la fonction d’attaque et de déplacement que nous verront par la suite.

II. Les fonctions initiales

Au lancement du jeu, deux choix s'offrent à l'utilisateur jouer une partie partie précédemment sauvegardée (en la chargeant) ou lancer une nouvelle partie, étudions d'abord le cas d'une nouvelle partie.

a. La fonction init()

Au lancement de la fonction init(), les positions initiales (fixes) des personnages des deux joueurs sont lus dans deux fichiers textes (déjà présents dans les fichiers du jeu) puis en se servant de ces coordonnées, elle place les personnages dans l'arène (après l'avoir vidée au préalable).

Puis, en commençant par le joueur1 les joueurs attribueront les 3 points de compétence par personnage et les nommant par la même occasion.

Entre outre les personnages des deux joueurs sont placés face à face (orientation nord pour les personnages du joueur 1 et sud pour les personnages du second joueur).

b. Le couple load/save

Ces deux fonctions sont intimement liées, en effet, à chaque fin de tour un appel de sauvegarde sera fait, puis à chaque début de partie on demande bien évidement au joueur s'il souhaite charger la partie.

Les fonctions sont relativement simple, on vérifie chaque cases de la matrice, si elle est vide la fonction sauvegarder stockera un 8 dans le fichier sauvegarde .txt, autrement elle stockera 1 pour signifier que c'est un personnage du joueur 1 et 2 pour le joueur 2, ensuite toutes les valeurs de la structure seront lues. Puis stockée. Le chargement reprendra le même ordre pour charger et initialiser une partie.

II. Le « cœur » du Jeu

a. Fonctions affichage(), gestion_pa, gest_pers et init_gest

Commençons simplement, la fonction afficher parcours l'arène et affiche la case en question en fonction de son type t-camp, permettant de déterminer si un personnage et trouve et , auquel cas l'orientation de la pièce ainsi que l'équipe à laquelle celle ci appartient sont affichés.

La fonction gest_pers permet de gérer les tours de chaque joueur en enregistrant les personnages morts à chaque tour, pour cela, elle appel init_gest qui stocke les coordonnées des personnages dans un tableau en ajoutant un indice (le numéro du personnage du joueur, ce qui lui permet de les distinguer et de déterminer si un personnage est mort ou non (pour passer directement au personnage suivant dans le jeu). Elle détermine également qui gagne et qui perd avec un printf.

b. La fonction gestion_pa()

Il n'est pas faux de dire que la fonction gestion_pa() est le cœur même de notre programme, c'est elle qui permet au joueur de contrôler leurs personnages, donnant à chaque tour 3 points d'actions à un personnage qui peux alors les utiliser pour tourner, avancer dans la direction choisi ou encore attaquer.

Ensuite par soucis du détail, lors du déplacement des personnages, le tableau Arena[x][y] est mise à jour mais la position d'où le personnage est parti soit juste la partie t_camp est modifiée (et mise à vide) afin de diminuer le travail de l'ordinateur.

En outre, c'est aussi la fonction gestion_pa() qui gère les points de vies des personnages qui, s'ils tombent à zéro, ou devienne négatifs, permet à gest_pers de reconnaître le personnage comme mort.

III. Implication personnelle

a. Répartition du travail

Pour ce qui est de la répartition du travail sur ce projet nous avons réparti les taches selon les préférences et facilités des membres du groupe.

De ce faite Elliot Candale s'est occupé des fonctions de sauvegarde et de chargement ainsi que de la gestion des tours c'est aussi à lui que nous devont l'interface graphique, basée sur la SDL (dans deux jours).

Puis Linus König a codé gestion_pa() avec les fonctions pour se déplacer, attaquer et se tourner, et Il s'est occupé également de l'affichage dans le terminal ainsi que de la fonction init().

Finalement pour ce qui est du rassemblement et du fonctionnement du programme final dans Battle_Arena.c ce fut plus une collaboration entre deux des membres du groupe.

b. Problèmes rencontrés

Un des problèmes principaux rencontré était la mise à jour des positions des personnages nécessaire pour l'utilisation de gest_pers() prenant en paramètre les coordonnées du personnage qui peut exécuter une action, les personnages ne se déplaçait donc qu'une seule fois et ne faisai rien d'autre.

Un autre problème était les cas « improbable » comme par exemple une pièce ce déplaçant en dehors de l'arène (problème qui nous avons finalement résolue et interprété comme un « suicide » du personnage), ou encore un utilisateur qui essaie de charger une partie alors qu'aucune partie ne fut sauvegarder auparavant(reste encore un problème d'ailleurs).

Finalement, un problème moins relié à la programmation était la gestion du temps, et comment le répartir mais nous verrons cette question plus en détail dans la partie suivante.

IV. Conclusion

a. Résultat final

Finalement nous sommes parvenus à créer un jeu de Battle Tactic jouable avec la possibilité de sauvegarder et charger sa partie, Le jeu jusque là semble dépourvu de bugs

b. Regrets/possibilités d'améliorations

Pour finir avec les regrets que nous éprouvons, par exemple de ne pas avoir pensé plus tôt à donner au joueur la possibilité de choisir quel personnage utiliseras les points d'actions donnés,

Ensuite nous avions pensé à un système de classe (guerrier, archer, mage etc...) que nous avons finalement laissé derrière nous par manque de temps, remplacé par le système de points de compétences actuel.

Enfin une autre possibilité d'améliorer « Battle Arena » aurait été une génération aléatoire de l'arène (ou choisie dans une liste d'arènes disponibles) avec des aléa de terrains tels que des montagnes augmentant la range mais prenant plus de points d'actions .

Pour conclure il y avais beaucoup de moyens d'améliorer notre jeu pour lui donner un charme plus personnel, nos seules limites restant notre imagination afin de continuer à faire vivre ce projet. Cependant nous sommes plutôt satisfait de notre rendu finale.

La chose principale que nous avons compris lors de ce projet est que la motivation et l'amour envers le sujet joue beaucoup sur le temps investi et est presque aussi importante que la capacité à coder elle même.