Le Mans Université

Licence Informatique *2ème année*

Conduite de projet
Tactics Arena

`https://github.com/ChevalierSoft/Mortal-Kombat-Arena.git`

AIT ATMANE Daris
EL KANDOUSSI Adnan
COLIN Kévin

19 avril 2019

Table des matières

1 Introduction

Notre projet a pour but de développer un jeu reprenant les bases du jeu de stratégie déjà existant Tactics arena.
C'est un jeu de stratégie qui se joue à deux joueurs en tour par tour, similaire aux échecs dans la forme, mais qui possède une dimension beaucoup plus stratégique que celui-ci.
C'est un jeu de stratégie en ligne gratuit qui se joue à deux joueurs en tour par tour.
Tactics arena premier du nom est sorti en 2003 (fermer le premier mai 2016) sur micro-ordinateur et développe par Vito .
Notre tactics arena aura donc une version terminal et une version graphique avec la SDL. En plus de cela, il y a la possibilite de jouer en réseau.

2 Organisation du travail

Au niveau de la répartition des tâches au sein du groupe, nous avons séparé le projet en trois grands aspects : l'environnement, les entités personnage et le déroulement du jeu. Cette séparation nous à semblé la plus pertinente au moment où nous nous étions penchés sur la question.

L'environnement regroupe les éléments visuels dans lesquels le/les joueur(s) évolueront pendant la partie.
Dans ce projet l'environnement est fait de deux manières différentes : une version sur terminal et une version graphique SDL.

Les entités personnage sont les unités présentes dans chacune des équipes des joueurs qu'ils peuvent contrôler. Chacun des personnages possède des caractéristiques et un panel de compétences.

Enfin le déroulement du jeu est la manière dont les parties se dérouleront. Cette partie regroupe la gestion des tours de jeu et le design des menus de choix des joueurs.

Voici une vue générale de notre répartition récapitulant les trois grandes tâches :

- EL KANDOUSSI Adnan - L'environnement SDL et terminal.
- COLIN Kevin - La création des structures et les attaques.
- AIT ATMANE Daris - Le déroulement du jeu.

Au fil du projet

3 Conception

3.1 Regle du jeu

C'est un jeu en un contre un. Les joueurs ont pour but de détruire toutes les unités de leur advesaire en effectuant des actions de déplacements ou d'attaques chacun leur tour.
Chaque unité possède des caractéristiques qui lui sont propre et surtout une palette de sort en fonction de sa classe.
La partie s'achève lorce qu'un des deux joueur n'a plus d'unité en vie.

3.2 Fonctionalites

Généralités :
Le programme doit prendre en compte l'état de chacun des pions après chaque action.
Si la vie d'un pion a atteint zéro, il est alors considérer comme mort, dans ce cas, son image est supprimé pour etre changer en petit crane du plateau et ces pv fixés à 0. Il ne doit plus être utilisable par le joueur propriétaire.
Si tous les pions d'un des joueurs sont mort, le programme s'arrête et affiche le vainqueur de la partie.
Les joueurs doivent toujours être capables de garder un œil sur le terrain après chacun de leur tour et durant leur tour.

Déplacements :
Le programme doit permettre de sélectionner un pion à déplacer et le déplacer dans toutes les directions possibles.
Un déplacement effectué doit pouvoir être annulé et le joueur doit pouvoir choisir d'attaquer ou de déplacer à nouveau son pion.

Attaques :
Un pion qui attaque doit avoir le choix entre différentes compétences ayant chacune une portée limitée.
Le pion qui attaque doit pouvoir choisir sa cible parmi celles situées à sa portée.
Si aucune cible n'est à portée alors le pion ne peut pas attaquer.

Réseaux :

Pour pouvoir jouer en réseaux l'un des deux joueurs doit choisir d'etre l'hôte dans le menu 'menu_online', et le second doit le rejoindre grâce au même menu.

Si l'un des joueurs perd la connection la partie s'arrête. Pour éviter toute triche, le fichier de sauvegarde des personnages du client est hashé en md5, et envoyé à l'hôte. L'hôte hash à son tour son fichier de sauvegarde, et le compare avec celui du client. Si les deux chaînes sont différentes, la partie ne se lance pas, et un message de sauvegarde éroné est envoyé aux joueurs.

4 Développement

4.1 Développement de la carte et sdl

Dans tactics arena, l'environnement est représentée par une 'carte' cadrillée sur laquelle les unités de chaque joueur évoluent.

Comme deux versions du jeux on été prévues, une version terminal et une version SDL, il est donc nécessaire de prévoir deux manières de créer cette carte.

Le choix qui est fait ici est de garder une structure commune aux deux versions affichées de la carte.
L'environnement global est donc composé de deux structures :

- une structure 'dalle' qui représente une case de la carte.
- une structure 'carte' qui comporte une matrice de taille définie d'éléments de type 'dalle', et deux entiers représentant le nombre de cases en x et en y de cette carte.

Chaque dalle possède un entier représentant son type de terrain, le 'land', ainsi qu'un pointeur de type 'personnage' sur le personnage présent sur cette case.
Les différents types de terrains sont les suivants :

- la terre
- l'eau
- le feu
- le vide
- la roche
- le vide (représente l'absence de case)

Au commencement de la partie la carte est initialisée, c'est à dire que chaque terrain correspond au vide et chaque pointeur sur personnage est mis à NULL.

La fonction 'charger partie' permet de creer une carte à partir de deux fichiers nommés 'save_m.txt' et 'save_p.txt'. Le premier servira à récupérer les données de terrain de la carte,et le second à récupérer les personnages et leurs coordonnées de case.

Pour ce qui est de l'affichage, il doit se faire de deux manières différentes en fonction du mode dans lequel le programme est lancé.
Si on lance le programme sans option, l'affichage se fera sur terminal. Si au lancement nous rajoutons l'option '-s', l'affichage sera sur SDL.

D'une part, pour l'affichage sur terminal, la fonction 'afficher_map' est appelée après un chargement réussi de la carte de jeu et une fois pendant chaque tour de jeu. Elle prend en paramètre un pointeur sur la carte chargée, la parcoure et affiche en printf deux matrices à l'écran, la première représentant les terrains, la seconde affichant les personnages.

Le terrain est affiché en tant qu'une matrice couleur. Chaque case est parcourue à l'aide d'une double boucle 'for' et à l'aide d'un 'switch' il est affiché la constante correspondante et la couleur du terrain sur cette case.
La deuxième matrice, celle qui affiche les personnages, sert à connaître après chaque fin de tour la position et l'état de chacune des unités.
Sur cette matrice, les personnages et les cases sont affichées à l'aide de symboles spéciaux :

- Les cases sont représentées par un symbole carré de couleur grise choisi arbitrairement.
- Les personnages ont chacun un symbole spécial, lesquels ont été attribués en fonction de leur classe, et qui permet au joueur de savoir quel type d'unité il est entrain de jouer. Ce symbole est changé en 'crâne' dès lors que ce personnage est vaincu.

D'une autre part,pour l'affichage graphique en SDL, ce sera cette fois la fonction 'jouer' qui sera appelée pendant chaque tour avec en paramètres le renderer, un pointeur sur le personnage actuellement entrain de jouer, et un pointeur de carte.
Cet affichage sur SDL utilise le même pointeur sur carte que l'affichage en mode terminal. Le même pointeur est donc passé en parametre de cette fonction.
Cette fonction sert à afficher graphiquement dans une fenêtre l'état du jeu et permet au joueur de faire une action avec l'unité active une fois chaque tour.
La fenêtre est divisée en deux parties : en haut la carte et en bas le menu.
Voici les informations que nous donne la carte :

- Chaque case est représentée par un carré de taille définie de pixels et possède une texture en fonction de son type de terrain.
- Si une case possède un pointeur sur personnage non nul, l'unité pointée sera affiché au centre de cette case avec une texture correspondant à sa classe.

Voici les informations que nous donne le menu:

- Au centre différents boutons qui permettent dans un premier temps de choisir si l'on veut attaquer ou se déplacer avec l'unité active. Dans un second temps selectionner la case où le personnage devra se déplacer, ou executer une attaque sur un personnage ennemi (ou pas) en selectionnant une attaque parmis celles qui s'affichent et en cliquant ensuite sur le personnage ciblé.
- A droite les informations concernant le personnage actif : son nom, son portrait et ses points de vie.
- Sur la gauche des messages indiquant à chaque fois qu'une attaque a été executée, l'unité qui a initié l'attaque, l'unité cible, le nom de l'attaque et l'effet qui a été produit(perte de points de vie,changement de statut,déplacement,etc.)

Ces actions, comme des clics de souris ou l'appuie d'une touche sur le clavier, sont gérées dans une boucle 'd'évènement' grâce à la fonction 'Wait_Event()' qui prends en paramètre une variable de type 'SDL_Event' qui représente une action quelconque de l'utilisateur.

Après l'appel de 'Wait_Event()', le programme attend une action du joueur, si une action précise est menée par le joueur et si cette action est un des 'events' dans le 'switch', le traitement à l'interieur du 'case' sera executé.

A chaque fin de tour, toutes les textures affichées sur la fenêtre sont effacées et réaffichée selon le nouvel état de la carte et des personnages.

4.2 Développement des entités personnages

Le jeu Tactics Arena repose essentiellement sur la gestion d'unités, la structure est l'élément principal du projet.

Le programme utilise donc deux structures.
Une structure appelée personnage_t qui contient les caractéristiques globale des unités :

- -Le nom de l'unité (nom)
- -Le symbole pour l'identifié (pp)
- -Les points de vie (pv)
- -Les points de magie (pm)
- -La position de l'unité sur le plateau en x (px)
- -La position de l'unité sur le plateau en y (py)
- -Les points d'intel (intel)
- -Les points de force (force)
- -Les points de chance (chance)
- -Le nombre de spell (nb_spell)
- -Le nom des spells (nom_spell)
- -Le tableau de spell (tab_spell)
- -L'état du personnage si il est en mouton (est sheep)
- -Les points de bouclie (est_shield)
- -Permet de voire si le personnage est vivant ou mort (est_mort)
- -Permet de voire si le personnage est empoisone (est_empoisone)
- -Permet de voire si le personnage est enfeu (est_enfeu)
- -Permet de voire si le personnage est aveugle (est_aveugle)
- -Permet de voire si le personnage a un enchantement actif et sur combien de tour (est_enchante)

Un switch 'classe', est appelée depuis la fonction init_hero et applique sur la structure globale de personnage_t les spells qui lui son propre en fonction de la classe choisie. La fonction init spell permet en plus d'appeler les différents spells, de copier le nom des spells et de mettre chacun d'entres eux dans un tableaux pour pouvoir appeler les fonctions d'attaques depuis celui ci. Après la création d'un personnage celui-ci est envoyé dans une liste qui stocke les personnages de tous les joueurs dont le délimiteur fait aussi partie. Cette liste permet notament de vérifier les différents états des personnages après chaque action sans avoir besoin de parcourir toute la map ! Enfin, une fois la

partie terminé la mémoire crée pour la liste des peronnages est libéré ainsi que toute les structures contenaient et enfin les tableaux contenant les nom des spells.

4.3 Dévelopement de la gestion des tours du jeu

Juste avant que la partie se lance, on prend l'information dans les fichiers de sauvegardes de la carte et des personnages. Cette initialisation fait on peut commencer.

La partie se déroule tant que les deux équipes on un personnage en vie. Les tours suivent la liste des personnages initialisés à la lecture de la sauvegarde. C'est à l'équipe 1 de jouer jusqu'au moment où l'on croise le délimiteur. Ce délimiteur est unique, de type personnage_t il sert à indiquer que c'est à l'équipe 2 de jouer jusqu'à la fin de la liste. Apres quoi, le programme boucle et se met en tete de liste. Ce sera à l'équipe 1 de jouer.
La condition d'arrêt de la partie est soit de capituler, soit qu'une des deux équipe n'ai plus de personnage en vie. Avant qu'un personnage puisse faire une action, on parcours la liste, et on regarde si les personnages ont une altération d'état(empoisonné, brulure, etc...) grâce à la fonction 'detection_etat'. Si c'est le cas on applique l'effet correspondant. Ceci fait on verifie que le personnage à qui c'est le tour n'est pas mort.
Ensuite c'est au joueur de faire une action. Dans le terminal on propose plusieurs menu donnant acces à des attaques, à la sauvegarde, au déplacement, etc... Quand le joueur à fait ses choix, on effectue la commande demandé en s'adaptant au choix faits, puis on passe au suivant dans la liste.

Lors d'une partie en réseau, les choix faits sont enregistrés dans une structure de type 'action_t', on a :

- L'id du personnage qui réalise l'action (id)
- Le choix d'attaquer, se déplacer etc... (choix_menu)
- Si le personnage attaque, il faut donner qu'elle attaque il a fait (numero_fonction)
- position sur l'axe x de l'attaque ou du déplacement (tx)
- position sur l'axe y de l'attaque ou du déplacement (ty)

Après avoir appliqué en local l'action faite, on l'envoie à l'autre joueur par la fonction 'send_action'. Pendant ce temps l'autre programme attend cette action puis l'applique grâce à 'reciv_action'

5 Résultat

La charge de travail étant dur à porter et le sujet n'interressant qu'en surface, peu d'etre nous se sont vraiment mit au travail. Nous n'avons pas pu assembler le réseau et l'affichage dans un même programme. L'éditeur de carte ne permet pas de placer des personnages.
Nous n'avons que peu de classes, sort, sprites opérationnels comparé à nos attentes. Il a aussi été impossible d'installer la librairie SDL et son module ttf sur nos machines malgré les multiples tentatives sur plusieur OS. Le tutoriel fourni n'ayant pas aidé. Cela à affecté le groupe et la cadance de travail. Nous orions pu lier les deux programmes mais le faire sans pouvoir compiler ne nous parait pas être une bonne idée. Cela dit, le programme fonctionne.

6 Conclusion

6.1 Améliorations

Des fonctionnalités qui auraient pu voir le jour dans ce projet :

- Le pathfinding, une animation de déplacement des unités graphique qui les feraient avancer d'un point A à un point B.
- La musique pour apporter une meilleur immersion dans la partie et donner une identité à notre jeu.
- Plus de spells et de mécanique de jeu pour complexifier les parties.
- Une IA avec différents mode de difficultés pour pouvoir s'entrainer seul.
- Rajouter un mode histoire

6.2 Apports

Ce projet nous a appris sur le travail en groupe et la programmation. Nous avons appris à contourner les limites de langage.

7 Annexes

lol :')