MORGENSTERN Emma p2110244
DJAMAKORZIAN Sasha p2101813

Cahier des charges

Nom du projet : LIF-FIGHTER

1. Présentation et description du projet

Notre projet consistera à imaginer et développer un jeu de combat 2D, similaire à Street Fighter, en C++ avec la bibliothèque graphique SDL2 et ses extensions.

Le jeu se déroule dans un environnement 2D où 2 joueurs s'affrontent en utilisant différentes techniques de combat. Le but du jeu est de vaincre l'adversaire en lui infligeant des dégâts jusqu'à ce que sa barre de vie soit épuisée, le gagnant gagne alors 1 point. Une partie se joue en BO5, c'est-à-dire lorsqu'un joueur à 3 points.

Chaque joueur a le choix entre différents personnages pour vaincre son adversaire. Chaque personnage a deux attaques de base, un coup de poing et un coup de pied. Le jeu pourra donc se jouer à 2, en 1 contre 1, ou bien tout seul pour tester les mouvements. A l'ouverture du jeu, un menu apparaît, il suffit de choisir le mode de jeu pour lancer une partie. Le jeu se déroule avec une musique de fond et différents bruitages en fonction des attaques.

2. Contraintes

- Le jeu sera développé en C/C++ sous Linux.
- Les librairies utilisées seront SDL2, SDL2_mixer et SDL2_image.
- Le code respectera le standard suivant : indentation, règle homogène de nommage des variables et fonctions, pré/post-conditions claires.
- Le code sera géré et archivé sur GitLab.
- La documentation du code sera produite par Doxygen.
- Un diagramme des classes permettra d'avoir une vision de haut niveau de l'implantation.
- Un diagramme de Gantt permettra de suivre l'avancement du projet.
- Les outils de debug et de profiling seront gdb et Valgrind.
- Le code sera présenté devant un jury lors d'une soutenance.

3. Listes des fonctionnalités

- Menu principal.
- Choix du mode de jeu.
- Différentes attaques, saut, accroupissement, défense.
- (Choix du personnage et sons.)

4. Déroulement du projet

Il y aura un test de régression dans chacun de nos modules.

Tâche 0 : Rédiger le cahier des charges et définir le diagrammes des classes.

Membres impliqués : Tous.

Durée : 1 semaine

Tâche 1 : Développement des fonctionnalités de base en mode texte

Durée : 3 semaines

Nous ne mettrons pas en place les animations en mode texte car nous considérons qu'elles font partie du mode graphique.

Tâche 1.0 : Créer les répertoires bin, src, obj, data et doc et importer sur le répertoire Git

Tâche 1.1 écriture et test du module Game (Game.h /.cpp)

Le module Game est défini par 2 Fighter, un état de jeu, un menu.

Cette classe va permettre de démarrer le jeu, quitter le jeu, commencer une partie et quitter la partie.

Nous allons mettre en place une classe Menu qui va permettre grâce à l'énumération 'MenuItem' de choisir ce que nous souhaitons effectuer (mode 2 joueurs, options du jeu, quitter celui-ci..)

Tâche 1.2: écriture et test du module Fighter (Fighter.h/cpp) et Action (Move.h)

Le Fighter est défini par un nom, un nombre de points de vie, une position X/Y, un état (booléen : alive) et une action qu'il va effectuer.

Le Fighter peut se déplacer vers la droite ou la gauche et peut effectuer des actions (sauter, s'accroupir, donner des coups et les bloquer).

Le fichier Move.h sera le fichier d'en-tête pour les actions du Fighter.

Tâche 1.3: écriture et test des fonctions précédentes (mainTXT.cpp)

Le mainTXT.cpp contiendra les tests sur les fonctions et procédures de Fighter et Game.

Tâche 1.4: configurer Doxygen et documenter le code (*.h)

Le fichier .doxy sera le fichier de configuration Doxygen.

Tâche 1.5: débugger avec Valgrind

Tâche 1.6 : écrire la commande make

Tâche 2 : Développement des fonctionnalités de base en mode graphique (avec SDL2)

Durée : 2 semaines.

Tâche 2.1 : mise en place des librairies SDL2 et SDL2_image

Prise en main des librairies.

Tâche 2.2 : écriture et test de la boucle de jeu en SDL2 dans le module GraphicMode (GraphicMode.h/.cpp)

Le module GraphicMode est défini par un affichage du module Game et d'une boucle pour les différents évènements.

Tâche 2.3 : écriture et test de l'affichage et des mouvements d'un Fighter dans la fenêtre graphique (mainGraphMode.cpp)

L'implémentation de l'utilisation des touches du clavier pour déplacer le Fighter et effectuer les mouvements.

Tâche 2.4: documenter le code avec Doxygen (*.h)

Tâche 2.5: débugger avec Valgrind

Tâche 2.6 : écrire la commande make

Tâche 3 : Ajout des mouvements du Fighter implémentés avec les animations associées

Durée : 2 semaines

Tâche 3.1 : écriture et test des fonctions autour de l'animation des Fighters (Sprite.h/.cpp, Animation.h/.cpp)

Le module Sprite est défini par une image, la hauteur et la largeur de l'image et une position.

Le module Animation est défini par un tableau de Sprite, un lapse de temps pour l'animation, un Sprite affiché (currentframe) , un numéro pour le Sprite affiché (frameIndex), un état de boucle (true si le Fighter ne bouge pas, false s'il se déplace) et un état de lancement de l'animation (true si le Fighter change d'action)

Tâche 3.2: écriture et test des fonctions des modules Sprites et Animation (mainAnim.cpp)

Vérification de la bonne implémentation des animations avec chaque mouvement associé pour le personnage

Tâche 3.3: documenter le code avec Doxygen (*.h)

Tâche 3.4: débugger avec Valgrind

Tâche 3.5 : écrire la commande make

Tâche 4 : Ajout des collisions des personnages et corrélation avec la perte de points de vie

Durée : 3 semaines.

Tâche 4.1 : écriture et test du module Hitbox (Hitbox.h/.cpp)

La Hitbox permet de déterminer les endroits où les collisions ont lieu. Un total de 4 Hitbox va être mis en place.

Ce module est défini par une position et une dimension (longueur et largeur.) que l'on va récupérer grâce aux procédures.

Une Hitbox sera dédiée pour l'attaque du personnage et les 3 autres, situées au niveau de la tête, du torse et des jambes seront créés.

Tâche 4.2 : écriture et test du module Collisions (Collisions.h/.cpp)

Le module Collisions est défini par une hitbox pour chaque personnage, deux Fighter, un état qui détermine si la collision a eu lieu et le numéro du Sprite affiché lors de la collision.

Tâche 4.3 : écriture et test de l'affichage de la classe RoundState (RoundState.h/.cpp)

La classe RoundState est définie par les points de vie des deux Fighters, des possibles Collisions ainsi que du nombre de rounds joués.

Il y a une mise en place de la barre de vie ainsi que des rounds effectués notamment les victoires du personnage de manière graphique en prenant en compte les collisions et la perte de points de vie. Les modifications de l'état du Round seront affichées grâce aux procédures Update() et Render().

Tâche 4.4: documenter le code avec Doxygen (*.h)

Tâche 4.5: débugger avec Valgrind

Tâche 4.6 : écrire la commande make

Tâche 5 : Ajout de sélection de personnages, le menu de démarrage et la musique (optionnel)

Durée : 2 semaines.

Tâche 5.1 : Implémentation de plusieurs autres personnages ainsi que d'un menu de sélection sur le même modèle que le personnage de base que nous avons codé.

Mise en place graphique du menu de démarrage du jeu.

Tâche 5.2: Implémentation de la musique en fond et si possible d'un son pour chaque action du Fighter grâce à la librairie SDL_mixer.

Tâche 5.3: Implémentation de combos ou d'autres mouvements afin de diversifier les actions des personnages.

Tâche 5.4: documenter le code avec Doxygen (*.h)

Tâche 5.5: débugger avec Valgrind

Tâche 5.6 : écrire la commande make

Tâche 6 : Finalisation du projet

Durée : 1 semaine.

Tâche 6.1 : vérification de tout le code (entêtes, #include, fonctions/procédures, etc) et modification des potentiels problèmes.

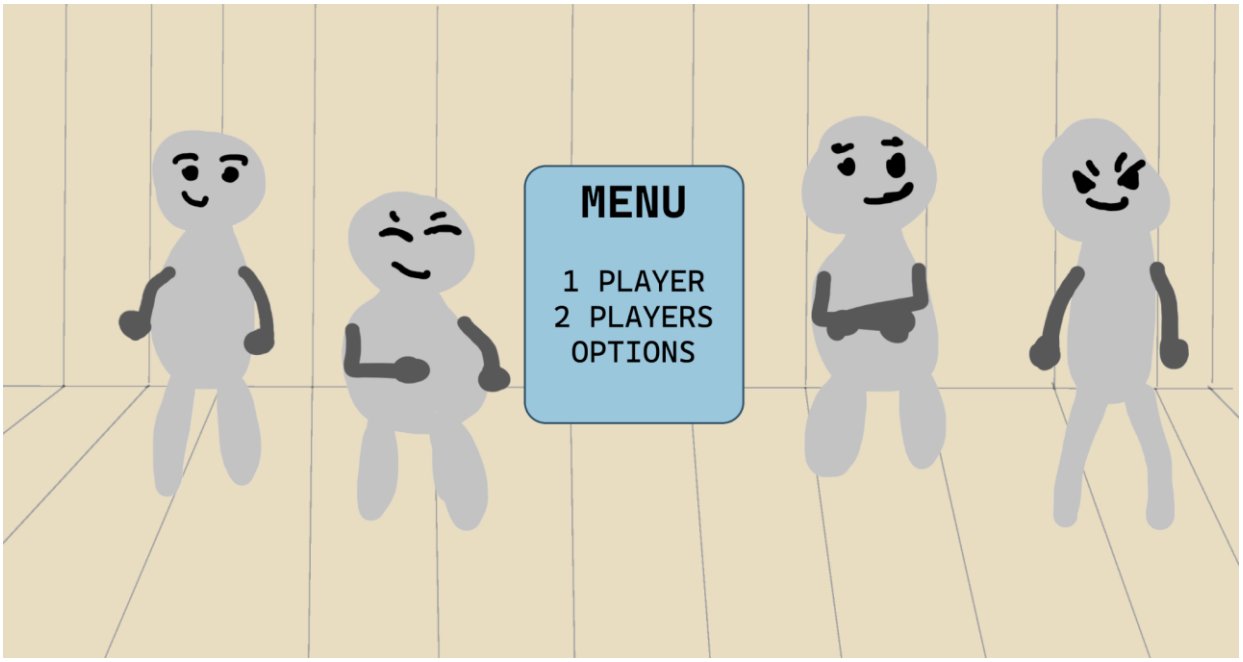
Tâche 6.2 : vérification de la documentation Doxygen

Tâche 6.3: vérification debug avec Valgrind

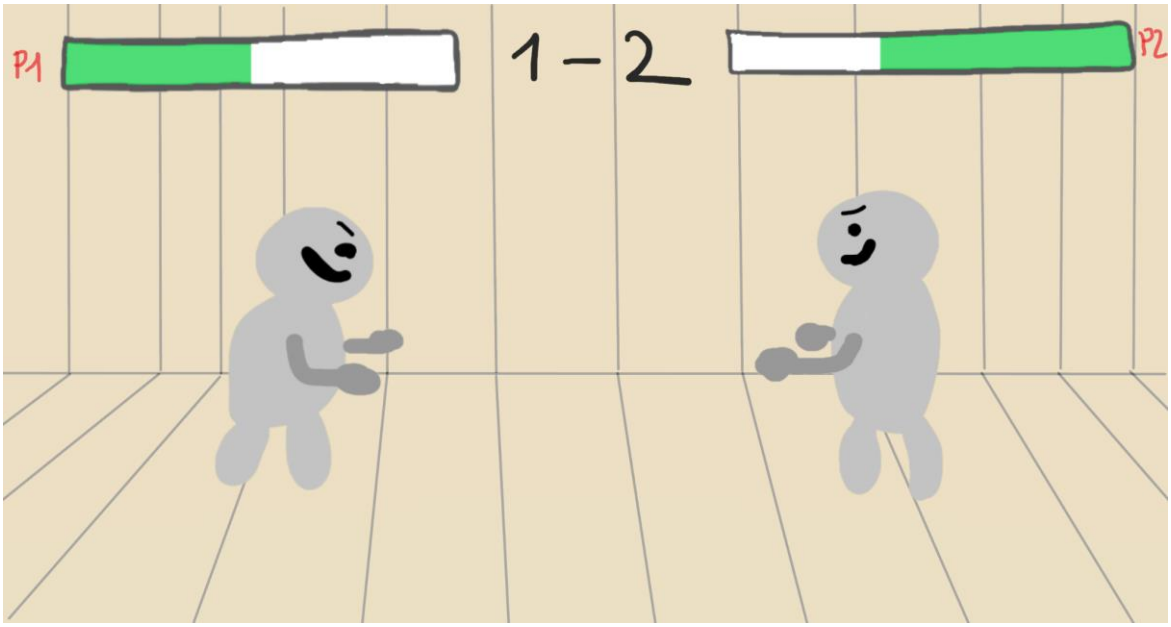
Tâche 6.4 : vérification du Makefile

5. Annexes

1.croquis du menu principal.



2.croquis d'une partie.



3.Diagramme de Gantt.

Tâches / Semaines	**28/02**	**7/03**	**14/03**	**16/03**	**21/03 demo**	**28/03**	**4/04**	**11/04**	**18/04**	**25/04**	**02/05**
Tâche 0											
Tâche 1 : TXT											
Tâche 1.0											
Tâche 1.1											
Tâche 1.2											
Tâche 1.3											
Tâche 1.4											
Tâche 1.5											
Tâche 1.6											
Tâche 2 : GRAPHIQUE											
Tâche 2.1											
Tâche 2.2											
Tâche 2.3											
Tâche 2.4											
Tâche 2.5											
Tâche 2.6											
Tâche 3 : ANIMATIONS											
Tâche 3.1											
Tâche 3.2											
Tâche 3.3											
Tâche 3.4											

Tâche 3.5											
Tâche 4 : COLLISIONS											
Tâche 4.1											
Tâche 4.2											
Tâche 4.3											
Tâche 4.4											
Tâche 4.5											
Tâche 4.6											
Tâche 5 : OPTIONNEL											
Tâche 5.1											
Tâche 5.2											
Tâche 5.3											
Tâche 5.4											
Tâche 5.5											
Tâche 5.6											
Tâche 6 : FINALISATION											
Tâche 6.1											
Tâche 6.2											
Tâche 6.3											
Tâche 6.4											
SOUTENANCE											

4.Diagrammes de classes UML de notre projet.