RAPPORT PROJET ARENE

Anis Hannachi, Francois-Xavier Hilaire, Thomas Sauwal,
Matthieu Eccher et Mael Adler

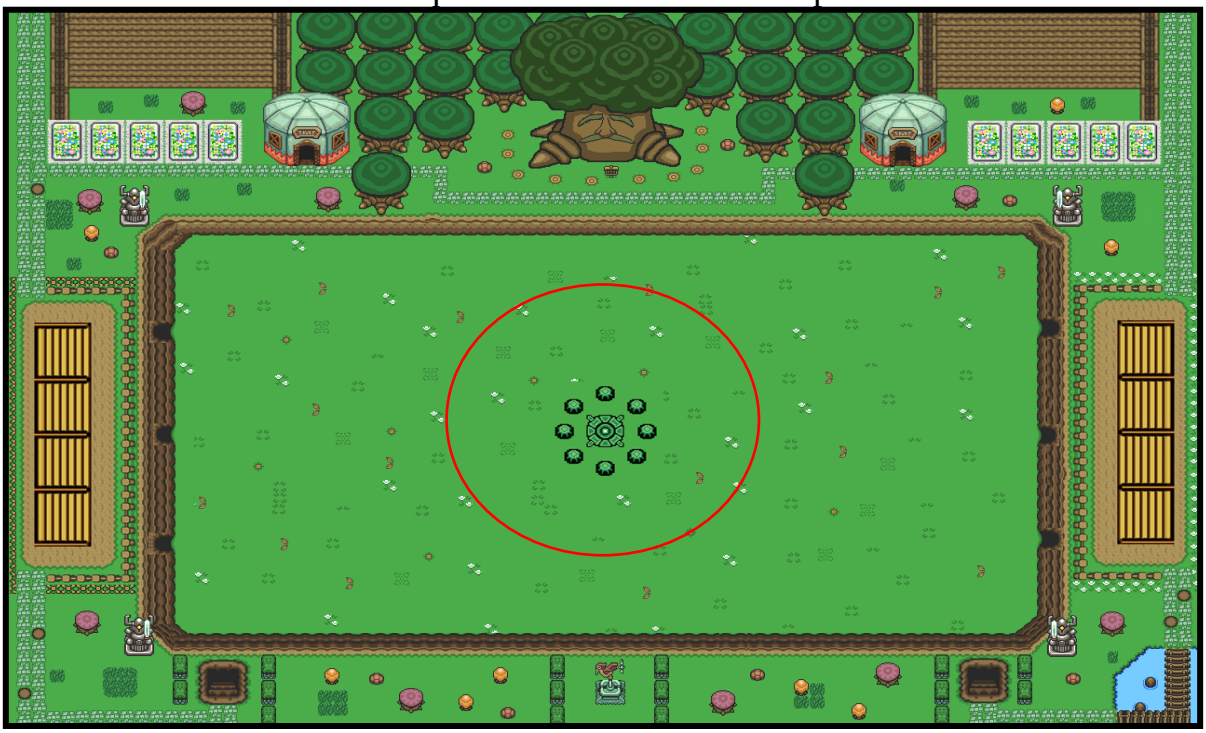
SOMMAIRE :

I) Règles de notre arène

Les principaux objectifs de ce projet sont de concevoir une arène. Cette arène doit être un tableau 2D qui doit permettre deux types de confrontation entre joueur : un combat en 1 contre 1 et une confrontation de 28 IA qui se battent en même temps dans l'arène. Nous avons donc décidé de partir sur une arène du thème de Zelda pouvant permettre à la fois des combats en 1 contre 1 et un combat simultané de 28 IA. L'ambiance globale de l'arène est plutôt joviale avec un décor varié, haut en couleurs et très fleuris. De nombreux spectateurs animés sont répartis de part et d'autre de l'arène dans des tribunes pour dynamiser l'arène. Comme dit précédemment nous sommes plongés au cœur de l'univers Zelda avec des personnages et des sprites tirés des jeux "A Link To The Past" et "The Legend Of Zelda Minish Cap". Le tout accompagné de musiques tirés de la série mais également de morceaux de symphonies.

1) Début de partie

Le début de partie se déroule comme suit : les joueurs apparaissent en cercle dans l'arène de façon à être tous à égal distance du centre. Tous les joueurs possèdent une base de rubis qui est la même pour tout le monde.



Les joueurs jouent dans un ordre déterminé aléatoirement et effectuent des actions qui seront décrites dans la seconde sous-partie.

2) Règles du jeu

Par la suite chaque joueur peut effectuer différentes actions : les actions basiques de déplacement (haut, bas, gauche, droite) ainsi que des actions spéciales (donner un coup d'épée) ainsi que des actions impliquant des items qui sont toujours en stade de développement (bombe, arc, parer un coup d'épée avec un bouclier). Les joueurs se déplacent selon un quadrillage. Un système d'anticollision permet d'empêcher deux joueurs d'être sur la même case. Chacune de ces actions ont pour but de permettre aux joueurs de s'entretuer (coup d'épée) ou de se déplacer pour ramasser des items. Les items sont les rubis qui pourront être ramassés au sol ou en détruisant d'autres items destructibles, les fameux pots du jeu Zelda.

3) La fin de partie

Comme évoqué précédemment le but du jeu est de ramasser le plus de rubis possible. Il y a différents types de rubis, de couleurs différentes, rapportant chacun un nombre de points différents. Néanmoins il y a plusieurs façons d'obtenir des points autre que de ramasser des rubis, de détruire des pots ou de frapper un autre joueur. Lorsqu'un joueur frappe un autre en lui donnant un coup d'épée celui-ci perd un certain nombre de rubis qui sont automatiquement attribuer au joueur ayant lancé l'attaque. De ce fait le vainqueur peut être le joueur avec le plus de rubis à la fin du temps imparti, mais pas seulement. En effet, de manière à équilibrer le jeu, une IA a été ajouté par nos soin. Son but est de poursuivre le joueur ayant le plus de points et de lui infliger des dégâts afin de lui faire perdre son avantage. A partir d'un certain temps une ombre démoniaque apparaît (l'ombre de Ganon) et n'aura, jusqu'à la fin de la partie, qu'un seul but... pourchasser le premier afin de rééquilibrer la compétition. Il faut voir ce mécanisme de jeu comme une « carapace bleu » de Mario kart. Le joueur ayant perdu ces rubis après les dégâts de l'IA n'est donc plus en première place. L'IA retourne donc au centre de l'arène et se mettra de nouveau à poursuivre le joueur en première position dans le classement.

II) Explication du code

Notre code se compose d'une fonction principale et de plusieurs sous fonctions qui exécutent chacune des actions précises sur l'arène comme par exemple les mouvements de Ganon (L'IA qui poursuis le joueur ayant le plus de rubis), les mouvements du joueur, des PNJ dans les gradins, mais aussi le déroulement du

jeu en lui-même. Nous décomposerons donc ces fonctions en 3 parties. Les fonctions qui impliquent des actions dans l'arène, les fonctions qui réalisent le jeu, et la fonction principale le main. Toutes ces fonctions sont regroupées dans un seul et même fichier (main.ccp) et disposent de tout leur prototypes ou headers dans un autre (constante.h). Les différentes fonctions de la sdl sont également présentent dans presque toute les fonctions (SDL_BlitSurface, SDL_Color, SDL_Event, SDL_SetColorKey, SDL_Surface,etc…).

1) La fonction principale, main

La fonction principale tient une place importante dans notre code. En effet, la fonction initialise tous les paramètres qui seront nécessaire pour la suite du code comme les coordonnées du fond d'écran, l'image du menu du jeu, l'icône de la fenêtre, les paramètre de la fenêtre d'affichage du jeu, les différentes musiques qui s'exécuteront au fil de la partie ainsi que la variable qui annonce les évènements.

2) Les fonctions réalisant le jeu (play, win, etc…)

Les fonctions réalisant le jeu sont divisées en deux parties. Il y a la fonction play et la fonction win. La fonction play permet de gérer tout ce qui touche de près ou de loin au déroulement du jeu : les personnages qui doivent être animé, l'initialisation du timer, les musiques et sons qui devront être jouées à des moments précis, une partie de la gestion des déplacements des joueurs, ainsi que l'apparition des rubis dans l'arène. La fonction win quant à elle intervient au niveau de la fin de partie. Rappelons qu'en fin de partie le gagnant est soit le joueur possédant le plus de rubis soit le dernier survivant. En fin de partie la fonction win affiche sur un écran de victoire le gagnant avec son nombre de points.

3) Les fonctions impliquant des actions dans l'arène

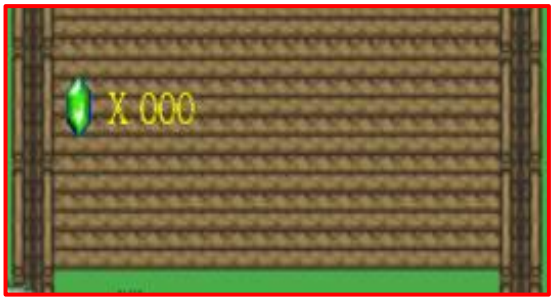
Notre code est équipé d'une multitude de fonctions qui permettent de réaliser des actions dans notre arène. Commençons par l'une des plus importantes, la fonction setup_pictures. Son rôle est déterminant dans le code car elle référence toutes les images de tous les PNJ et personnages jouables de l'arène ainsi que toutes les positions qu'ils peuvent avoir (de face, de profile de dos). Chaque personnage est représenté par un tableau dont les valeurs sont les positions qu'ils peuvent avoir.

```
void setup_pictures (SDL_Surface *link[6],SDL_Surface *rupees[3], SDL_Surface *ganon[5], SDL_Surface *zelda[6], 
{


                /* IMAGE LINK */
    link[UP] = IMG_Load("link_up.bmp");
    SDL_SetColorKey(link[UP], SDL_SRCCOLORKEY, SDL_MapRGB((*link)->format, 0, 0, 255));
    link[DOWN] = IMG_Load("link_down.bmp");
    SDL_SetColorKey(link[DOWN], SDL_SRCCOLORKEY, SDL_MapRGB((*link)->format, 0, 0, 255));
    link[RIGHT] = IMG_Load("link_right.bmp");
    SDL_SetColorKey(link[RIGHT], SDL_SRCCOLORKEY, SDL_MapRGB((*link)->format, 0, 0, 255));
    link[LEFT] = IMG_Load("link_left.bmp");
    SDL_SetColorKey(link[LEFT], SDL_SRCCOLORKEY, SDL_MapRGB((*link)->format, 0, 0, 255));
//     link[HIT] = IMG_Load("link_hit.bmp");
  //  SDL_SetColorKey(link[HIT], SDL_SRCCOLORKEY, SDL_MapRGB((*link)->format, 0, 0, 255));
    //link[DOWN_ANIM] = IMG_Load("link_down_floor.bmp");
    //SDL_SetColorKey(link[DOWN_ANIM], SDL_SRCCOLORKEY, SDL_MapRGB((*link)->format, 0, 0, 255));


                /* IMAGE RUBIS */
    rupees[GREEN_RUPEE] = IMG_Load("green_rupee.bmp");
    SDL_SetColorKey(rupees[GREEN_RUPEE], SDL_SRCCOLORKEY, SDL_MapRGB((*rupees)->format, 0, 0, 255));
    rupees[BLUE_RUPEE] = IMG_Load("blue_rupee.bmp");
    SDL_SetColorKey(rupees[BLUE_RUPEE], SDL_SRCCOLORKEY, SDL_MapRGB((*rupees)->format, 0, 0, 255));
    rupees[RED_RUPEE] = IMG_Load("red_rupee.bmp");
    SDL_SetColorKey(rupees[RED_RUPEE], SDL_SRCCOLORKEY, SDL_MapRGB((*rupees)->format, 0, 0, 255));

                /* IMAGE GANON */
    ganon[UP] = IMG_Load("ganon_V6.bmp");
    SDL_SetColorKey(ganon[UP], SDL_SRCCOLORKEY, SDL_MapRGB((*ganon)->format, 0, 0, 255));
    ganon[DOWN] = IMG_Load("ganon_V6.bmp");
    SDL_SetColorKey(ganon[DOWN], SDL_SRCCOLORKEY, SDL_MapRGB((*ganon)->format, 0, 0, 255));
    ganon[LEFT] = IMG_Load("ganon_V6.bmp");
    SDL_SetColorKey(ganon[LEFT], SDL_SRCCOLORKEY, SDL_MapRGB((*ganon)->format, 0, 0, 255));
    ganon[RIGHT] = IMG_Load("ganon_V6.bmp");
    SDL_SetColorKey(ganon[RIGHT], SDL_SRCCOLORKEY, SDL_MapRGB((*ganon)->format, 0, 0, 255));
    //ganon[FIGHT] = IMG_Load("ganon_V6.bmp");
    //SDL_SetColorKey(ganon[], SDL_SRCCOLORKEY, SDL_MapRGB((*ganon)->format, 0, 0, 255));
```

Puis viens la fonction setup_map qui définit à partir de l'image les bords de l'arène et permet de faire apparaitre les rubis sur des cases de façon aléatoire.

Ensuite vient la fonction timer qui comme son nom l'indique affiche un chronomètre mais aussi le score du joueur.

```
void timer (char temps[20],char score[6],int time,int lastTime,int stime,int mtime,int points)
{
    if (stime < 10)
    {
        sprintf(temps, "Timer : %d : 0%d",mtime,stime);
    }
    else
    {
        sprintf(temps, "Timer : %d : %d",mtime,stime);
    }
    if (points < 10)
    {
        sprintf(score, "X 00%d",points);
    }
    else if (points < 100)
    {
        sprintf(score, "X 0%d",points);
    }
    else
    {
        sprintf(score, "X %d",points);
    }
}
```

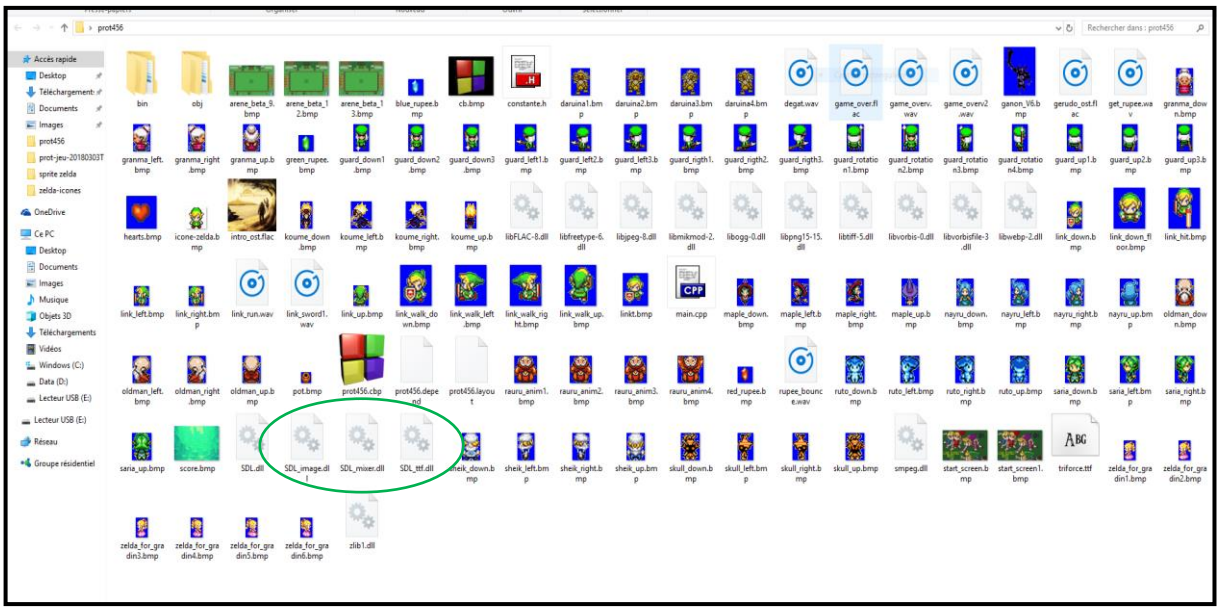


Maintenant viennent les fonctions moove_player et ganon_moove qui permettent le déplacement des joueurs et de l'ombre de Ganon (IA qui poursuis le joueur ayant le plus de rubis). Ces fonctions prennent en compte les bordures de l'arène et la fonction moove_player est dotée d'un test de collisions avec les rubis sur les vases qui sont contiguës. De plus, cette dernière déduit les points d'un joueur en fonction du type de rubis qu'il a ramassé ce qui est utile pour que Ganon poursuive le premier joueur de la partie, avec la fonction ganon_moove.

Enfin, viennent les fonctions animation et garde qui ont pour but d'animer les personnages non joueurs qui se situent dans les gradins et les gardes qui sont en mouvement tout autour de l'arène

III) Prérequis pour l'exécution du code

Notre code utilise plusieurs facettes de la SDL qui nécessiteront d'être installées afin de pouvoir profiter de l'arène dans les moindres détails. En effet, en plus de la SDL classique nous avons travaillé sur notre code avec SDL mixer (pour la musique), SDL ttf (pour insérer du texte) et SDL image (pour les graphismes). De plus il faut intégrer toutes les images présentent dans le jeu dans le dossier du projet (sprites des personnages, arène, musique, police d'écriture etc…) Bien évidement nous avons utilisé CodeBlocks, un IDE pour coder en C.


prot456
Rechercher dans : prot456
Accès rapide
Desktop
Téléchargement
Documents
Images
prot456
prot-jeu-20180303T
sprite zelda
zelda-icones
OneDrive
Ce PC
Desktop
Documents
Images
Musique
Objets 3D
Téléchargements
Vidéos
Windows (C:)
Data (D:)
Lecteur USB (E:)
Lecteur USB (E:)
Réseau
Groupe résidentiel
bin
obj
arene_beta_9.bmp
arene_beta_12.bmp
arene_beta_13.bmp
blue_rupee.bmp
cb.bmp
constante.h
daruina1.bmp
daruina2.bmp
daruina3.bmp
daruina4.bmp
degat.wav
game_over.flac
game_overv.wav
game_overv2.wav
ganon_V6.bmp
geruldo_ost.flac
get_rupee.wav
granma_down.bmp
granma_left.bmp
granma_right.bmp
granma_up.bmp
green_rupee.bmp
guard_down1.bmp
guard_down2.bmp
guard_down3.bmp
guard_left1.bmp
guard_left2.bmp
guard_left3.bmp
guard_rigth1.bmp
guard_rigth2.bmp
guard_rigth3.bmp
guard_rotation1.bmp
guard_rotation2.bmp
guard_rotation3.bmp
guard_rotation4.bmp
guard_up1.bmp
guard_up2.bmp
guard_up3.bmp
hearts.bmp
icone-zelda.bmp
intro_ost.flac
koume_down.bmp
koume_left.bmp
koume_right.bmp
koume_up.bmp
libFLAC-8.dll
libfreetype-6.dll
libjpeg-8.dll
libmikmod-2.dll
libogg-0.dll
libpng15-15.dll
libtiff-5.dll
libvorbis-0.dll
libvorbisfile-3.dll
libwebp-2.dll
link_down.bmp
link_down_floor.bmp
link_hit.bmp
link_left.bmp
link_right.bmp
link_run.wav
link_sword1.wav
link_up.bmp
link_walk_down.bmp
link_walk_left.bmp
link_walk_right.bmp
link_walk_up.bmp
linkt.bmp
main.cpp
maple_down.bmp
maple_left.bmp
maple_right.bmp
maple_up.bmp
nayru_down.bmp
nayru_left.bmp
nayru_right.bmp
nayru_up.bmp
oldman_down.bmp
oldman_left.bmp
oldman_right.bmp
oldman_up.bmp
pot.bmp
prot456.cbp
prot456.depend
prot456.layout
rauru_anim1.bmp
rauru_anim2.bmp
rauru_anim3.bmp
rauru_anim4.bmp
red_rupee.bmp
rupee_bounce.wav
ruto_down.bmp
ruto_left.bmp
ruto_right.bmp
ruto_up.bmp
saria_down.bmp
saria_left.bmp
saria_right.bmp
saria_up.bmp
score.bmp
SDL.dll
SDL_image.dll
SDL_mixer.dll
SDL_ttf.dll
sheik_down.bmp
sheik_left.bmp
sheik_right.bmp
sheik_up.bmp
skull_down.bmp
skull_left.bmp
skull_right.bmp
skull_up.bmp
smpeg.dll
start_screen.bmp
start_screen1.bmp
triforce.ttf
zelda_for_gradin1.bmp
zelda_for_gradin2.bmp
zelda_for_gradin3.bmp
zelda_for_gradin4.bmp
zelda_for_gradin5.bmp
zelda_for_gradin6.bmp
zlib1.dll