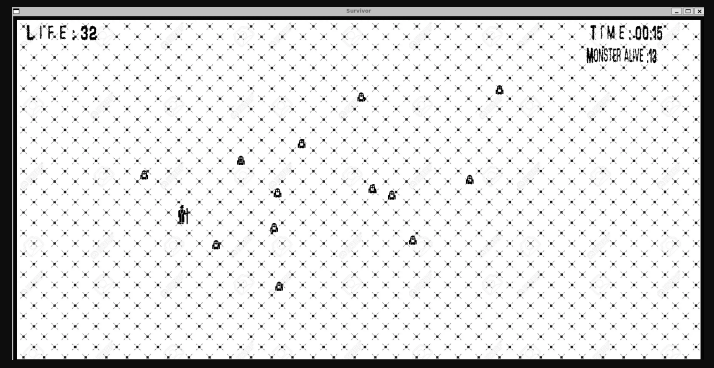
PROJECT LIFAPCD

Survivor

Réalisé par : - KANE Ousmane
- NEDJAR Amine
- DIA Abdou-Karim

Petit jeu de survie, le but étant de survivre le plus longtemps possible à une horde de monstre venant de toute les directions !!



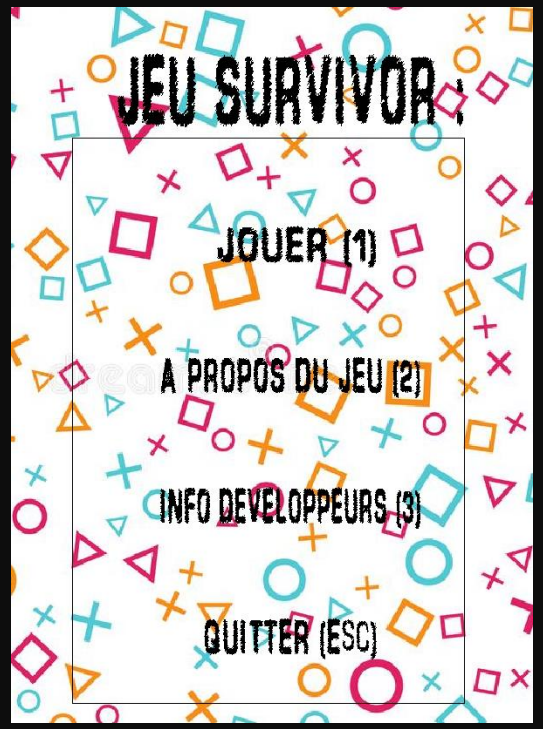
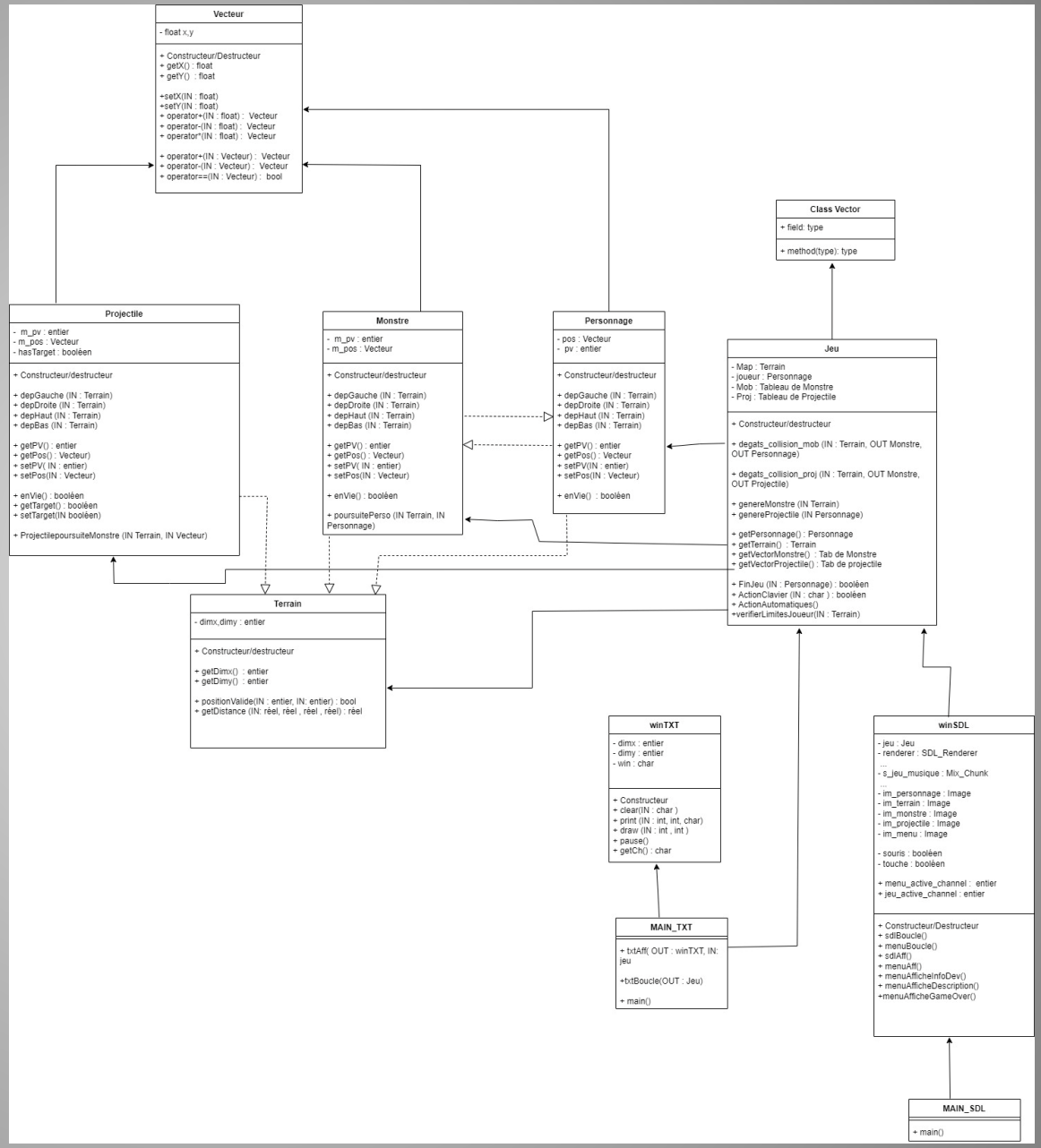


Diagramme des classes



Module Jeu.h/.cpp

- Regroupe tous nos modules et permet des actions entre-elles …

```
#include "Terrain.h"
#include "Personnage.h"
#include "Monstre.h"
#include "Projectile.h"
#include "vector"

class Jeu {
private:
    Terrain map;
    Personnage joueur;
    vector<Projectile> proj;
    vector<Monstre> mob;
```

- … et les manipules par le biais de plusieurs fonctions :

```
Jeu();
/** @brief Constructeur par défaut **/

~Jeu();
/** @brief Destructeur par défaut **/

Personnage getPersonnage() const;
/** @brief recupere le Personnage **/

Terrain getTerrain() const;
/** @brief recupere le Terrain **/

Monstre getMonstre() const;
/** @brief recupere le Monstre **/

vector<Monstre> getVectorMonstre() const;
/** @brief recupere le tableau dynamique de monstre */

vector<Projectile> getVectorProjectile() const;
/** @brief recupere le tableau dynamique de projectile*/

void degats_collision_mob(const Terrain &map, Monstre &mob, Personnage &joueur);
/** @brief Procédure qui gère les degats infliger entre un différente entité **/

void degats_collision_proj(const Terrain &map, Monstre &mob, Projectile &proj);

void genereMonstre(const Terrain &map);
/** @brief Genere un monstre aleatoirement sur le terrain **/

void genereProjectile(const Personnage &joueur);
/** @brief genere un Projectile sur la coordonnée du personnage **/

bool FinJeu (const Personnage &joueur);
/** @brief Test si le jeu est terminer ou non **/


bool actionClavier(const char touche);
/** @brief action clavier */

void actionAutomatiques();
/** @brief action automatique, notamment déplacement auto des monstres et tire projectile*/

void verifierLimitesJoueur(const Terrain &map);
```

Module Jeu.h/.cpp

- Diverses **actions automatiques** via la fonction « jeu::actionAutomatiques » contenant :
  - ***La génération*** des Monstres et des Projectiles via :

```cpp
void Jeu::genereMonstre(const Terrain &map)
{
    // Génération d'un monstre avec x pv aleatoirement sur la map en fin de tableau
    mob.push_back(Monstre(Vecteur(rand() % map.getDimx(), rand() % map.getDimy()), 1));
}

void Jeu::genereProjectile(const Personnage &joueur)
{
    // Génération d'un projectile sur la position du joueur avec une vitesse par défaut
    proj.push_back(Projectile(joueur.getPos(), Vecteur(), 1));
}
```

  - ***La gestion des collisions*** entre monstre-personnage et projectile-monstre via :

```cpp
void Jeu::degats_collision_mob(const Terrain &map, Monstre &mob, Personnage &joueur)
{ // monstre & perso

    int new_pv_mob;
    int new_pv_joueur;

    int degats = rand() % 6 + 6; //Dégats aléatoire entre 6 et 12 pour le personnage

    // Calcul de la distance entre le personnage et le monstre
    float distance = map.getDistance(joueur.getPos().getX(), joueur.getPos().getY(), mob.getPos().getX(), mob.getPos().getY());

    // Si la distance entre le monstre et le personnage est <= 20 on inflige des degats
    if (distance <= 20)
    {
        new_pv_joueur = joueur.getPV() - degats; // joueur prend x points de degats par mob
        joueur.setPV(new_pv_joueur);

        new_pv_mob = mob.getPV() - 1; // mob prend x points de degats
        mob.setPV(new_pv_mob);
    }
}


void Jeu::degats_collision_proj(const Terrain &map, Monstre &mob, Projectile &proj)
{ // monstre & proj

    int new_pv_mob, new_pv_proj;
    // Si la distance entre le monstre et le projectil est <= 15 on inflige des degats
    if (abs( proj.getpos().getX() - mob.getPos().getX()) <=5 && abs( proj.getpos().getY() - mob.getPos().getY()<=5)) //15
    {
        new_pv_mob = mob.getPV() - 1; // monstre prend x points de degats par projectile
        mob.setPV(new_pv_mob);

        new_pv_proj = proj.getPV() - 1; // proj prend x points de degats par monstre
        proj.setPV(new_pv_proj);
    }
}
```

  - ***La gestion des déplacements*** par les monstres et tires des projectiles via :

```cpp
void Jeu::actionAutomatiques()
{
    // Parcourir les monstres
    for (auto it_m = mob.begin(); it_m != mob.end();)
    {
        // Vérifier si le monstre est mort et le supprimer s'il est mort
        if (!it_m->enVie())
        {
            it_m = mob.erase(it_m);
        }
        else
        {
            // Faire poursuivre le personnage par le monstre
            it_m->poursuitePerso(map, joueur);
            // Infliger des dégâts au personnage en cas de collision
            degats_collision_mob(map, *it_m, joueur);
```

```cpp
// Si un projectile est trouvé, envoyer le projectile sur le monstre
if (index_monstre_proche != -1)
{

    // Envoyer le projectile sur le monstre
        Vecteur cible = it_m->getPos();
        //Une boucle pour que la cible reste fixe, et tant qu'il n'est pas mort il coninue
            proj[index_monstre_proche].ProjectilePoursuiteMonstre(cible, map);
            proj[index_monstre_proche].setTarget(true); // Marquer le projectile comme ayant une cible
            proj[index_monstre_proche].setTarget(false); // Mettre hasTarget à false
            // Infliger des dégâts au monstre en cas de collision
            degats_collision_proj(map, *it_m, proj[index_monstre_proche]);

}
```

Module Jeu.h/.cpp

- Les **actions claviers** par la fonction « jeu::ActionClavier »
  - Notamment **les déplacements claviers** du joueurs

```
bool Jeu::actionClavier(const char touche)
{
    switch (touche)
    {
    case 'z':
        joueur.depHaut(map);
        break;

    case 'd':
        joueur.depDroite(map);
        break;

    case 'q':
        joueur.depGauche(map);
        break;

    case 's':
        joueur.depBas(map);
        break;

    }

    return false;
}
```

- **Test** pour le bon déroulement du jeu :
  - Un **test de fin de jeu** via :

```
bool Jeu::FinJeu(const Personnage &joueur)
{
    return (joueur.enVie());
}
```

  - Un test pour limiter les actions en fonction du Terrain via :

```
void Jeu::verifierLimitesJoueur(const Terrain &map)
{
    // Vérifier si le joueur a atteint les limites de l'écran
    if (joueur.getPos().getX() >= map.getDimx() && joueur.getPos().getY() >= map.getDimy()/3 && joueur.getPos().getY() <= map.getDimy()*2/3) {
        // Si le joueur atteint le bord droit, le faire réapparaître dans le tunnel gauche
        joueur.setPos(Vecteur(1, joueur.getPos().getY()));
    }
    else if (joueur.getPos().getX() <= 0 && joueur.getPos().getY() >= map.getDimy()/3 && joueur.getPos().getY() <= map.getDimy()*2/3) {
        // Si le joueur atteint le bord gauche, le faire réapparaître dans le tunnel droit
        joueur.setPos(Vecteur(map.getDimx()-1, joueur.getPos().getY()));
    }
    else if (joueur.getPos().getY() >= map.getDimy() && joueur.getPos().getX() >= map.getDimx()/3 && joueur.getPos().getX() <= map.getDimx()*2/3)
        // Si le joueur atteint le bord bas, le faire réapparaître dans la zone supérieure
        joueur.setPos(Vecteur(joueur.getPos().getX(), 1));
    }
    else if (joueur.getPos().getY() <= 0 && joueur.getPos().getX() >= map.getDimx()/3 && joueur.getPos().getX() <= map.getDimx()*2/3) {
        // Si le joueur atteint le bord haut, le faire réapparaître dans la zone inférieure
        joueur.setPos(Vecteur(joueur.getPos().getX(), map.getDimy()-1));
    }
    else if (joueur.getPos().getY() >= map.getDimy() && joueur.getPos().getX() < map.getDimx()/3) {
        // Si le joueur atteint le bord bas gauche, le faire réapparaître dans la zone supérieure droite
        joueur.setPos(Vecteur(map.getDimx()+1, map.getDimy()/3));
    }
    else if (joueur.getPos().getY() <= 0 && joueur.getPos().getX() < map.getDimx()/3) {
        // Si le joueur atteint le bord haut gauche, le faire réapparaître dans la zone inférieure droite
        joueur.setPos(Vecteur(map.getDimx()+1, map.getDimy()*2/3));
    }
    else if (joueur.getPos().getY() >= map.getDimy() && joueur.getPos().getX() > map.getDimx()*2/3) {
        // Si le joueur atteint le bord bas droit, le faire réapparaître dans la zone supérieure gauche
        joueur.setPos(Vecteur(1, map.getDimy()/3));
    }
    else if (joueur.getPos().getY() <= 0 && joueur.getPos().getX() > map.getDimx()*2/3) {
        // Si le joueur atteint le bord haut droit, le faire réapparaître dans la zone inférieure gauche
        joueur.setPos(Vecteur(1, map.getDimy()*2/3));
    }
}
```

Module Monstre.h/cpp

- Le **déplacement** d'un monstre via :
  - 4 fonctions pour chaque directions :

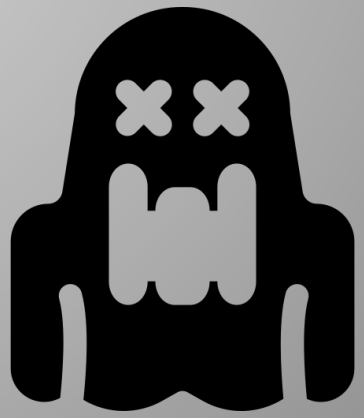
```
void Monstre::depDroite(const Terrain &t) {

    if (t.positionValide(m_pos.getX(),m_pos.getY()))
    {
        m_pos.setX(m_pos.getX() + DIST);
    }
}
void Monstre::depHaut(const Terrain &t) {
    if (t.positionValide(m_pos.getX(),m_pos.getY()))
    {
        m_pos.setY(m_pos.getY() - DIST);
    }
}
```

  - Une fonction permettant de ***suivre*** le joueur :

```
void Monstre::poursuitePerso(const Terrain& t, const Personnage& p) {
    if (!t.positionValide(m_pos.getX(), m_pos.getY())) {
    return;
    }
    // Obtenir la position du joueur et du monstre
    const int mX = m_pos.getX();
    const int mY = m_pos.getY();
    const int pX = p.getPos().getX();
    const int pY = p.getPos().getY();

    // Si le monstre et le joueur sont sur la même colonne ou ligne, se déplacer dans la direction correspondante
    if (pX == mX) {
        if (pY < mY) {
            depBas(t);  // Inverser la direction
        } else if (pY > mY) {
            depHaut(t);  // Inverser la direction
        }
        return;
    }
    if (pY == mY) {
        if (pX < mX) {
            depDroite(t);  // Inverser la direction
        } else if (pX > mX) {
            depGauche(t);  // Inverser la direction
        }
        return;
    }
```

- **L'état** du monstre (si il est en vie ou non) via :

```
bool Monstre::enVie() const{
    if (m_pv==0){
        return false;
    }
    else return true;
}
```



Module Projectile.h/cpp

- Le **déplacement** d'un projectile via :
  - 4 fonctions pour chaque directions :

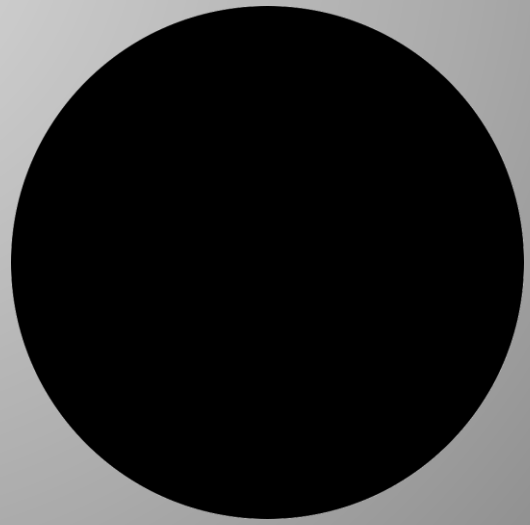
```
void Projectile::depDroite(const Terrain &t)
{
    if (t.positionValide(p.getX(),p.getY()))
    {
        p.setX(p.getX() + PR_DIST);
    }
}
void Projectile::depGauche(const Terrain &t)
{
    if (t.positionValide(p.getX(),p.getY()))
    {
        p.setX(p.getX() - PR_DIST);
    }
}
```

  - Une fonction permettant de cibler une cible :

```
void Projectile::ProjectilePoursuiteMonstre(const Vecteur& arrivee, const Terrain &t)
{
    if (p.getX() != arrivee.getX() || p.getY() != arrivee.getY()) {
        if (p.getX() > arrivee.getX()) {
            depGauche(t);
        } else if (p.getX() < arrivee.getX()) {
            depDroite(t);
        }
        if (p.getY() > arrivee.getY()) {
            depHaut(t);
        } else if (p.getY() < arrivee.getY()) {
            depBas(t);
        }
    }
}
```

- L'état du projectile (si il est en vie ou non) via :

```
bool Projectile::enVie() const{

    return (p_pv!=0) ;
}
```



Module winSdl.h/.cpp

- Les **affichages de notre Menu** de jeu :

```
////////////////////////////////////////////////////////////////////////////
/////////////////////////////FONCTIONS MEMBRES MENU ///////////////////////////
////////////////////////////////////////////////////////////////////////////
void SDLSimple::menuAff () {···

void SDLSimple::menuAfficheInfoDev() {···

void SDLSimple::menuAfficheDescription() {···

void SDLSimple::menuAfficheGameOver() {···
```

- Avec **la boucle** du Menu :

```
void SDLSimple::menuBoucle ()  {
    SDL_Event events;
    bool quit_menu = false;

    while (!quit_menu) {

        if (withSound && !Mix_Playing(menu_active_channel)) { //Pour résoudre le problème du son, on
        menu_active_channel= Mix_PlayChannel(-1,s_menu_musique,0); //Musique menu
        }
        while (SDL_PollEvent(&events)) {
            if (events.type == SDL_QUIT) quit_menu = true;           // Si l'utilisateur a clique sur
            else if (events.type == SDL_KEYDOWN) {                   // Si une touche est enfoncee
                    switch (events.key.keysym.scancode) {
                        case SDL_SCANCODE_1:
                            Mix_HaltChannel(menu_active_channel);   //Arreter la musique du menu
                            Mix_PlayChannel(-1,s_choix_menu,0);     // son des touches
                            sleep(1);
                            if (withSound) Mix_PlayChannel(-1,s_start_effect,0);
                            if (withSound) jeu_active_channel= Mix_PlayChannel(-1,s_jeu_musique,0);
                            sdlBoucle();
                            sleep(1);
                            break;
                        case SDL_SCANCODE_2: Mix_PlayChannel(-1,s_choix_menu,0);
                              menuAfficheDescription();
                            break;
                        case SDL_SCANCODE_3: Mix_PlayChannel(-1,s_choix_menu,0);
                            menuAfficheInfoDev();
                            break;
                        case SDL_SCANCODE_ESCAPE:
                            quit_menu = true;
                            break;
                            default: break;
                    }
                }
            }
        menuAff();
    }
    SDL_RenderPresent(renderer);
}
```

- Des **affichages complémentaires** pour notre jeu *:*
  - Tel quel le temps ou encore le nombre de Monstre vivant

```
// Afficher le timer :
        // Calculer le temps écoulé en minutes et secondes
        int elapsed_time = (SDL_GetTicks() - start_time) / 1000; // temps écoulé en secondes
        minutes = elapsed_time / 60;
        seconds = elapsed_time % 60;
        // Mettre à jour la chaîne de caractères pour afficher le temps
        sprintf(timer_string, " T i m e : %02d:%02d", minutes, seconds);
        // Créer la surface de texte et la charger dans la texture
        font_im.setSurface(TTF_RenderText_Solid(font, timer_string, font_color));
        font_im.loadFromCurrentSurface(renderer);
        SDL_Rect positionTimer; // position du texte sur l'écran
        positionTimer.x = 1500; positionTimer.y = 20; positionTimer.w =200;positionTimer.h = 50;
        SDL_RenderCopy(renderer, font_im.getTexture(), nullptr, &positionTimer);

// Afficher le nombre de Monstre :
        int nb_mob = jeu.getVectorMonstre().size();
        sprintf(number_string, "Monster alive :%d  " ,nb_mob);
        font_im.setSurface(TTF_RenderText_Solid(font, number_string, font_color));
        font_im.loadFromCurrentSurface(renderer);
        SDL_Rect positionNumber; // position du texte sur l'écran
        positionNumber.x = 1500; positionNumber.y = 80; positionNumber.w =200;positionNumber.h = 50;
        SDL_RenderCopy(renderer, font_im.getTexture(), nullptr, &positionNumber);

    // on permute les deux buffers (cette fonction ne doit se faire qu'une seule fois dans la boucle)
    SDL_RenderPresent(renderer);
}
```

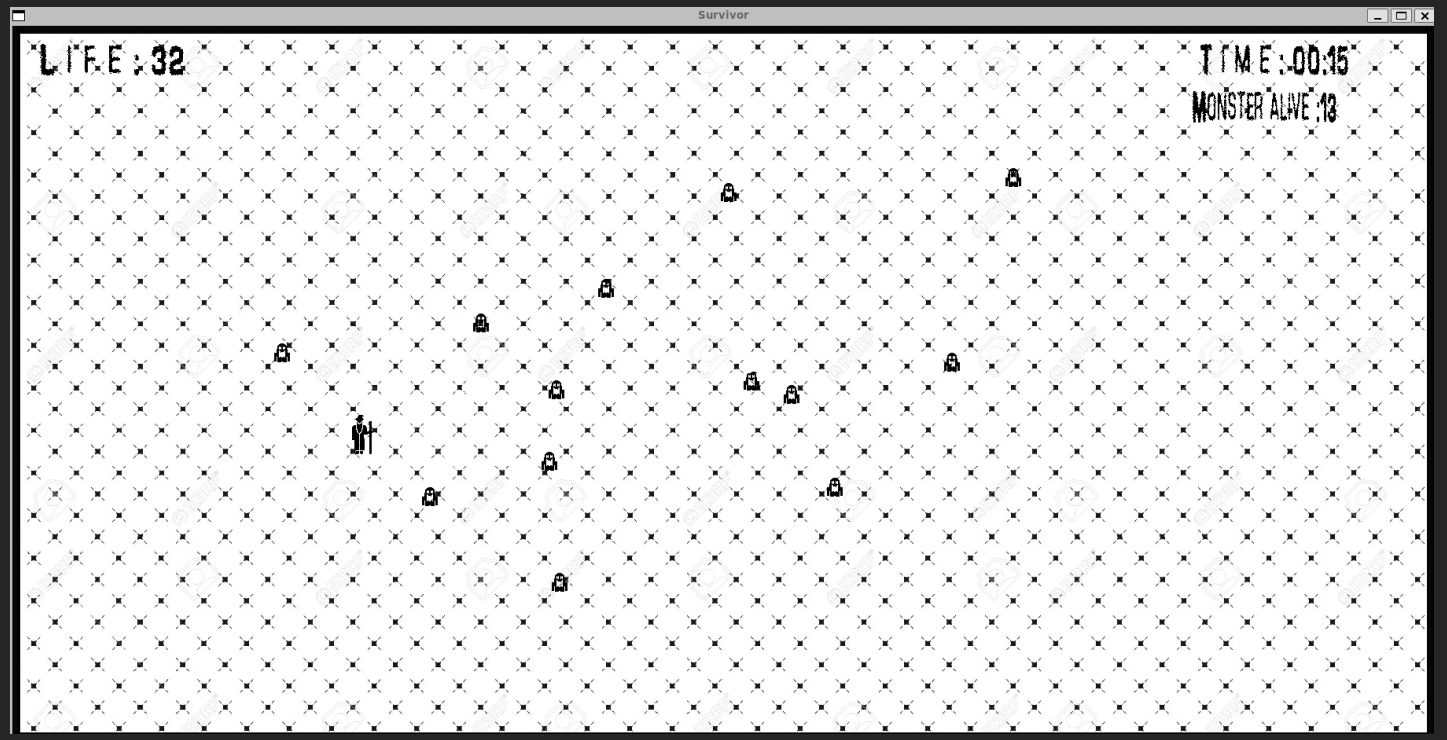
Module winSdl.h/.cpp

- ***Et la grande boucle de Jeu SDL :***

```
void SDLSimple::sdlBoucle () {
    SDL_Event events;
    bool quit_jeu = false;

    Uint32 t = SDL_GetTicks(), nt;
    Uint32 t_auto, t_monstre, t_projectile;
    t_auto = t_monstre = t_projectile = t;

    srand(time(NULL));
    int start_time = SDL_GetTicks(); // temps en millisecondes depuis le lancement du programme
    char timer_string[50]; // chaîne de caractères pour stocker le temps
    char number_string[50];
    int minutes = 0;
    int seconds = 0;


    // tant que ce n'est pas la fin ...
    while (!quit_jeu) {
/*...
        nt = SDL_GetTicks();
        if (nt-t_auto>40) {
            jeu.actionAutomatiques();
            t_auto = nt;
        }

        jeu.verifierLimitesJoueur(jeu.getTerrain()); // Verifier la position du personnage pour le placer dans l'écran

        if (nt-t_monstre>=1000){ // On ajout un monstre chaque seconde
            t_monstre=nt;
            jeu.genereMonstre(jeu.getTerrain());
        }
        if (nt-t_projectile>=3000){ // On ajoute un projectile chaque 3 secondes
            t_projectile=nt;
           jeu.genereProjectile(jeu.getPersonnage());
            Mix_PlayChannel(-1,s_tire_projectile,0);
        }
```

```
// tant qu'il y a des évenements à traiter (cette boucle n'est pas bloquante)
while (SDL_PollEvent(&events)) {
    if (events.type == SDL_QUIT) quit_jeu= true;           // Si l'utilisateur a cli
    else if (events.type == SDL_KEYDOWN) {              // Si une touche est enfonce
        switch (events.key.keysym.scancode) {
        case SDL_SCANCODE_UP:
            jeu.actionClavier('z');
            break;
        case SDL_SCANCODE_DOWN:
            jeu.actionClavier('s');
            break;
        case SDL_SCANCODE_LEFT:
            jeu.actionClavier('q');
            break;
        case SDL_SCANCODE_RIGHT:
            jeu.actionClavier('d');
            break;
        case SDL_SCANCODE_SPACE:
            jeu.actionClavier('o');
            break;

        case SDL_SCANCODE_ESCAPE:
            Mix_HaltChannel(jeu_active_channel);   //Arreter la musique du jeu
            quit_jeu = true;
            break;
        default: break;
        }
    }
}

if(!jeu.FinJeu(jeu.getPersonnage())) {
    Mix_PlayChannel(-1,s_fin_jeu,0);
    quit_jeu = true;
    sleep(1);
    menuAfficheGameOver();
}
// on affiche le jeu sur le buffer cach
sdlAff();
```

Conclusion



Merci pour votre écoute