RAPPORT SAE 1.02

Développement d'un space invader en C++

TABLE DES MATIERES

Jules Volpeï Clément Dugourd Andy Gonzales Noémie Djerian Elisée Leydier

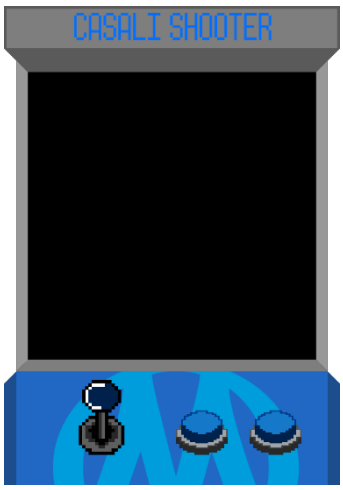
INTRODUCTION :

Lors de cet SAE 1.02 Nous avions comme projet de réaliser un space invader, nous avions pour obligation de faire un jeu en saisie de touche non canonique, cela pouvait se faire depuis le terminal ou graphiquement. Il devait y avoir un fichier de configuration yaml et une documentation doxygen, nous étions libres d'implémenter des règles ou de faire le design que l'on voulait. Dans le fichier **build** vous pourrez trouver tout ce qu'il faudra mettre dans le fichier de compilation (notamment les sprites, les fichiers txt ainsi que la musique).

REFLEXION AUTOUR DU SUJET :

Nous avons décidé de faire un jeu graphique car cela apporter des nouveautés, une nouvelle façon de programmer et cela nous a mis de devoir travailler sur la lecture d'une documentation, chose que nous serons amenés à refaire.

Qui dit jeu graphique dit Sprite, il a fallu réfléchir sur un thème général de ce space invader pour qu'il soit unique, quoi de mieux qu'un space invader sur notre bien aimé professeur d'initiation au développement.



Nous sommes donc venus avec l'idée du *CASALI SHOOTER*. Tous les sprites ont été fait par nous, ainsi que la musique.

Nous avons eu comme idée de représenter la fenêtre comme une réelle borne d'arcade avec la zone de jeu au milieu.

Comme particularité notable, nous pouvons noter la présence d'un menu.

LE MENU

Le menu est composé de 4 boutons :

Le bouton « **start** » va permettre de lancer le jeu mais avant de lancer le jeu, un nom vous sera demandé d'écrire, il va permettre par la suite de mettre à jour le scoreboard.

Le bouton « **scoreboard** » va permettre d'afficher un tableau des scores, ce tableau de score est généré grâce à un fichier contenant à chaque ligne : NOM␣SCORE, à l'aide d'une structure et de la fonction « **sort »** de la bibliothèque « **algorithm** » nous allons pouvoir placer le fichier dans un vecteur de structure qui va ensuite être trié grâce à sort pour être ensuite affiché.

```
void showScore(MinGL & window)
{
    ifstream data_file("score.txt");
    vector<playersStruct> playerScore;
    playersStruct playerStruc;
    while (data_file >> playerStruc.player >> playerStruc.point) { //Place every elements of the file in the structure
        playerScore.emplace_back(playerStruc);
    }
    sort (playerScore.begin (), playerScore.end(), isBetter); // Sort all players by their score
    vector<string> stringScore;
    if (playerScore.size()>10) { //Make that the structure only retains the 10 best players
        playerScore.resize(10);
    }
    for (size_t i = 0; i<playerScore.size(); ++i) { //Place every score and player on the screen
        stringScore.push_back(to_string(playerScore[i].point)); //Transform the score of the players into String so i can use nsGui::Text to show them
        window << nsGui::Text(Vec2D(190, 294+(30*i)), playerScore[i].player, KWhite, nsGui::GlutFont::BITMAP_9_BY_15)
               << nsGui::Text(Vec2D(525, 294+(30*i)), stringScore[i], KWhite, nsGui::GlutFont::BITMAP_9_BY_15);
    }
}
```

Le bouton « **options** » sert seulement pour le moment à changer entre le thème sombre et le thème clair du jeu. Mais il pourrait par la suite par exemple changer les contrôles ou le volume du son dans une prochaine mise à jour du jeu.

Le bouton « **quit** » comme son nom l'indique permet de quitter le jeu.

LA GESTION DES ENNEMIS

Chaque type d'ennemis va être représenté par une structure composée d'un booléen renvoyant son état de vie, d'un entier indiquant la direction du mouvement de l'ennemi et d'un vecteur de Sprite.

```
struct enemyStruct {
    std::vector<nsGui::Sprite> vecSprite;
    int rightOrLeft;
    std::vector<bool> state;
    void update(MinGL &window){
        for (unsigned i = 0; i < vecSprite.size(); ++i) {
            if (state[i]){
                window << vecSprite[i];
            }
        }
    }
};
```

Ce vecteur de Sprite sera généré par une fonction « genereVecSprite » permettant aussi de placer tous les ennemis au bon endroit dans la zone de jeu. Chaque ennemi sera généré de la sorte.

Les ennemis vont aller soit à droite soit à gauche, changeant de direction à chaque fois qu'il touche les bords de l'écran mais descendant petit à petit vers le joueur. L'ennemi va pouvoir tirer des torpédos.

```
void moveVecSprite(enemyStruct &vecSprite, string &playerLifeString, string &nameStr, Sprite &backgroundNoScreen, Sprite &creditSprite, MinGL &window){
    // If the sprites at the end do not touch the edges, move all the sprites at the same time
    if (vecSprite.vecSprite[0].getPosition().getX() < (600-64+50) ||
        vecSprite.vecSprite[vecSprite.vecSprite.size() - 1].getPosition().getX() > 0+50){
        for(Sprite &sprite : vecSprite.vecSprite){
            moveSprite(sprite, vecSprite.rightOrLeft *5, 0);
        }
    }
    // If the sprites at the end touch the edges, change direction and move down the sprites by 10 pixels
    if(vecSprite.vecSprite[vecSprite.vecSprite.size() - 1].getPosition().getX() > (600-64+50) && vecSprite.rightOrLeft == 1){
        vecSprite.rightOrLeft = -1;
        for(Sprite &sprite : vecSprite.vecSprite){
            moveSprite(sprite, 0, 10);
        }
    }else if(vecSprite.vecSprite[0].getPosition().getX() < 0+50 && vecSprite.rightOrLeft == -1){
        vecSprite.rightOrLeft = 1;
        for(Sprite &sprite : vecSprite.vecSprite){
            moveSprite(sprite, 0, 10);
        }
    }
    // if the sprite vector reached the bottom, GAME OVER
    if(vecSprite.vecSprite[0].getPosition().getY()>(600)){
        addScore(playerLifeString, nameStr);
        credit(window,backgroundNoScreen,creditSprite);
        exit(0);
    }
}
```

LA GESTION DU JOUEUR

Le joueur possède 3 vies et va pouvoir seulement bouger de gauche à droite, son but sera de tuer tous les ennemis avant qu'ils arrivent sur lui ou avant qu'il se fasse toucher 3 fois.

Afin de les tuer le joueur pourra aussi tirer des missiles, cependant nous avons rencontré un problème lors de la gestion des collisions. Une fonction « allDead » permettra de savoir si tous les ennemis sont mort à ce moment le plus maléfique des boss apparaitra (il ne faut pas prononcer son nom) ...

```
bool allDead(const enemyStruct & PPs) {
    for (size_t i = 0; i < PPs.state.size(); ++i) {
        //If the ennemy is alive
        if (PPs.state[i] == true) {
            return false;
        }
    }
    //There is no enemy alive anymore
    return true;
}
```

LA GESTION DES COLLISIONS

Nous avons trouvé qu'il existait une fonction nommée isCollide dans MinGL cependant nous n'avons jamais réellement réussi à la faire marcher, nous avons donc décidé de refaire la fonction nous-même, la fonction va vérifier si le missile ou par exemple le torpédo ennemis se situe entre le premier coin du joueur ou de l'ennemi et entre le second coin du joueur ou de l'ennemi, s'il l'est-il renverra vrai, le programme se chargera de soit faire perdre une vie au joueur, soit de faire disparaitre l'ennemi qui a été touché.

```
bool isTouching (const Vec2D firstCorner, const Vec2D secondCorner, const Vec2D test){
    return ((test.getX() <= secondCorner.getX() && test.getY() <= secondCorner.getY()) && (test.getX() >= firstCorner.getX() && test.getY() >= firstCorner.getY()));
}
```

LES INSTRUCTIONS DE JEU

Le jeu étant infini il n'y a pas de réel moyen de gagner il suffit de faire le plus gros score pour prendre la première place du tableau des scores, cependant à chaque vague les ennemis réapparaitront là où il était avant leur mort, évitant donc une partie infinie et des scores phénoménaux.

LES COMMANDES :

DANS LES MENUS :

Vous pourrez naviguer dans chaque menu à l'aide de **z** et de **s** pour descendre et monter, ainsi que **a** et **z** pour sélectionner soit le thème sombre ou le thème clair dans le menu **options,** pour sélectionner un bouton il y aura la touche « **ENTREE** » et revenir en arrière avec la touche « **ECHAP** ».

DANS LE JEU :

Vous pourrez vous déplacer à l'aide des touches **q** et **d** afin d'aller à gauche et à droite avec le mug et vous pourrez tirer un missile à l'aide de la touche « **ESPACE** », la touche « **ECHAP** » est toujours disponible pour revenir en arrière, cela gardera quand même votre progression en jeu.

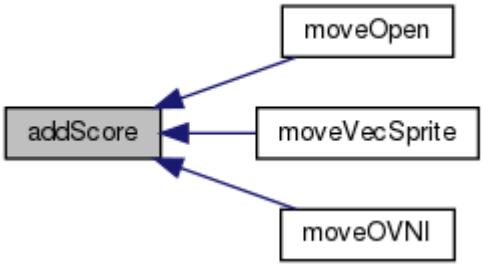
YAML

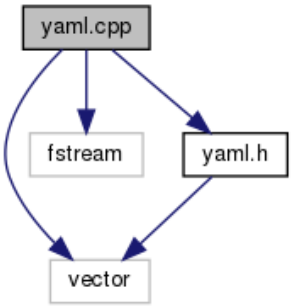
Les touches ne vous conviennent pas ? Il suffit d'accéder au fichier de configuration yaml, celui-ci vous permettra de changer n'importe quelle touche du jeu, faites cependant attention les paramètres sont des codes ascii si vous voulez changer les touches, il faudra regarder la touche correspondante en ASCII.

```
KBack : 27
KConfirm : 13
KLeftMenu : 97
KRightMenu : 122
KMovingDownMenu : 115
KMovingUpMenu : 122
KMovingLeftGame : 113
KMovingRightGame : 100
KShootGame : 32
```

LA DOCUMENTATION

Casali Shooter possède une documentation dans le dossier **html,** dans cette documentation vous pourrez trouver toutes les informations des structures ainsi que toutes les fonctions utilisées, des graphes pour les fonctions et pour les dépendances inclues seront disponibles pour une compréhension aisée.





Cette documentation possède dans la page d'accueil quelque instructions sur par exemple les touches de bases du jeu et comment les modifier.

My Project

Main Page | Classes | Files

Casali Shooter the game

instructions

This note provides some useful instructions for a good use of the game.* It's splited in two parts : the first one is about the menu and the last one is concerning the game itself.*

Menu

The following part contains all the informations needeed for a proper use of the menu.

z <– Allows you to move in the menu. Pressing this key will direct you to the button above.

s <– Allows you to move in the menu. Pressing this key will directo you to the button at the bottom.

ENTER <– Allows you to select the button at your current position in the menu.

ECHAP <– Allows you to comeback to the preview window once you cliked on a button.

Game