Rapport création de jeu 2D

Moteur : Cocos2d-x C++

Salma Nidar

Bakkali Ayoub

Table des matières

I. Problématique rencontré en développement :

Dans le développement de notre jeu Cocos2dx en C++, nous avons rencontré plusieurs difficultés. Tout d'abord, la création du jeu avec Tiled a été un défi en soi. Tiled est un outil de création de niveaux puissant, mais il nécessite une bonne compréhension de son interface et de ses fonctionnalités pour être utilisé efficacement (Jeu de Tuiles – GID – objets …). Nous avons dû consacrer du temps à apprendre à utiliser Tiled et à créer des niveaux de jeu qui étaient à la fois esthétiquement agréables et fonctionnellement solides.

Ensuite, nous avons eu du mal à implémenter le mouvement du joueur au clavier. Bien que cela puisse sembler être une tâche relativement simple, il y avait beaucoup de détails à prendre en compte pour que le mouvement soit fluide et réactif. Nous avons dû travailler sur les paramètres de vitesse et de friction du joueur, ainsi que sur la manière de gérer les collisions avec les bords de l'écran et les différents obstacles du niveau.

De plus, la détection des collisions a également été un défi pour nous. Nous avons dû déterminer comment détecter efficacement les collisions entre le joueur et les autres éléments du jeu, tels que les ennemis et les pièges. Nous avons dû également trouver un moyen de gérer les différents types de collisions de manière appropriée.

Aussi nous avons rentrés des difficultés liées a l'ajout de la physique à chaque niveau du jeu, il y a eu des problèmes pour synchroniser correctement la physique du joueur et de choisir les bonnes constantes pour un mouvement fluide (Restitution – Friction – masse – Gravité …). De plus, il été difficile de gérer les interactions entre le joueur et les éléments de la map, comme les obstacles.

En fin de compte, nous avons réussi à surmonter ces difficultés grâce à de nombreuses heures de développement et de tests minutieux.

II. Conception :

Le projet Pico Park consiste sur 7 class `MainMenu`, `LevelsMenu`, `Level1`, `Level2`, `Level3`, `Scene GameOver` et `Win`**.** L'utilisateur peut choisir dans **MainMenu** Scene de quitter ou d'aller LevelMenu scene. Ensuite choisir l'un des 3 niveaux ou de retourner vers **MainMenu**.
Si le joueur réussit en Level1 ou Level2 ou Level3 il passe vers Win Scene sinon vers GameOver Scene dont il aura le choix de rejouer ou de quitter vers MainMenu Scene.

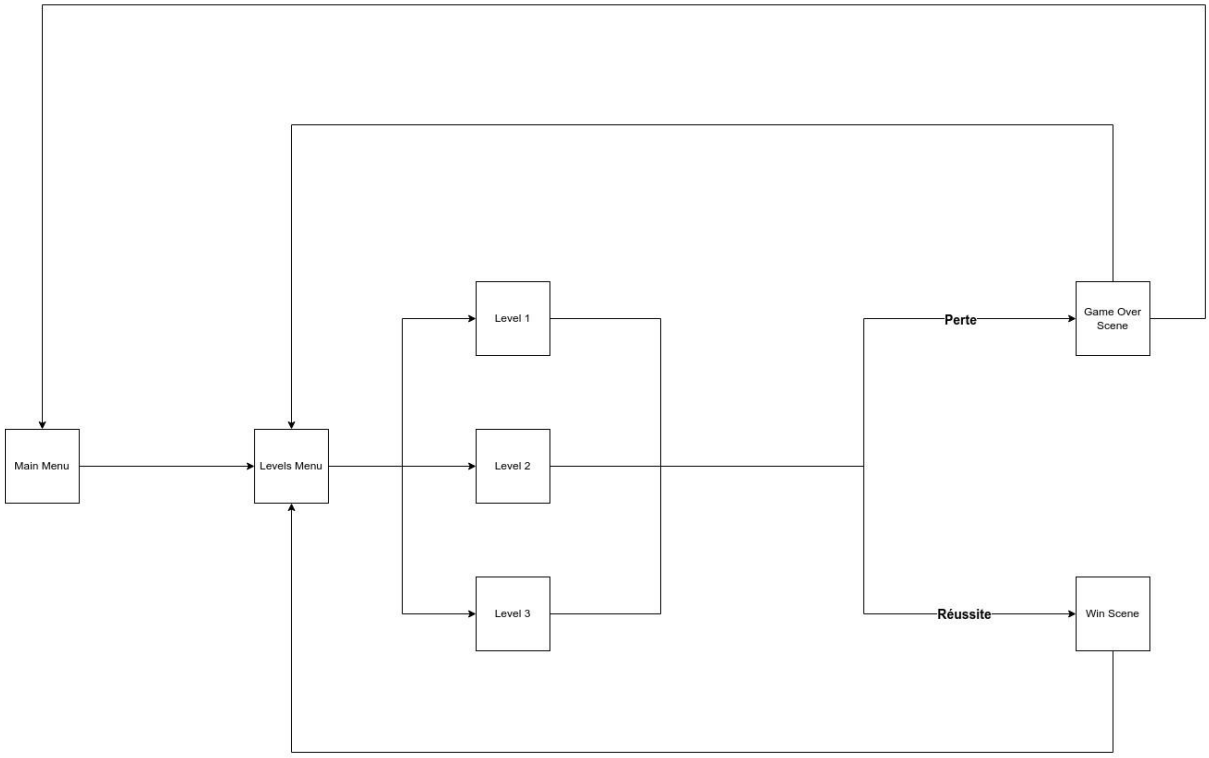

Figure 1 : Diagramme illustrant la conception du jeu

Chaque Class créé est hérité de la class `Scene` qui contient un macro `CREATE_FUNC` et une fonction `virtual bool init ()` où il y a l'implémentation du code du jeu.

Les Levels **s**ont créés par le logiciel Tiled, qui permet de créer des niveaux pour les jeux vidéo 2D, il offre une interface graphique conviviale pour la création de cartes en utilisant des tuiles prédéfinies ou des images personnalisées.

Remarque : un macro en C++ est une directive de préprocesseur qui remplace un mot-clé ou une expression par une autre chaîne de caractère lors de la compilation du code.

III. Main Menu Scene :

La classe `MainMenu`, représente le menu principal du jeu.

La méthode `init()` commence par charger une image de fond et le logo du jeu par les afficher à l'écran en utilisant la classe `Sprite`. Elle calcule ensuite un facteur de mise à l'échelle pour adapter l'image de fond à la taille de l'écran, en utilisant les dimensions de l'écran et les dimensions de l'image.

```
//Ajouter un background
auto background = Sprite::create("backgroung.png");

//Positionnement du Background au centre
background->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2) + origin.y));
this->addChild(background);

//calcule de factor pour mettre le background selon la taille de l'ecran
float rX = visibleSize.width / background->getContentSize().width;
float rY = visibleSize.height / background->getContentSize().height;

background->setScaleX(rX);
background->setScaleY(rY);

//Ajouter a sprite (Logo of the Game)
LogoPiccopark = Sprite::create("Piccopark Logo.png");
LogoPiccopark->setPosition(Vec2(235, 220));
this->addChild(LogoPiccopark);
```

Ensuite un menu en utilisant la classe `Menu` et en ajoutant deux éléments de menu, représentés par des images, avec les classes `MenuItemImage`. Ces éléments de menu correspondent respectivement au bouton "Start" et au bouton "Exit". Le code définit également la position de ces boutons à l'écran.

```
//adding a menu
auto menu_item_1 = MenuItemImage::create("Start button.png", "Start button.png", CC_CALLBACK_1(MainMenu::GotToLevelsMenu, this));
auto menu_item_2 = MenuItemImage::create("Exit Botton.png", "Exit Botton.png", CC_CALLBACK_1(MainMenu::menuCloseCallback, this));

//Positionement du menu Start et Exit
menu_item_1->setPosition(Vec2(235, 165));
menu_item_2->setPosition(Vec2(235, 135));

auto* menu = Menu::create(menu_item_1, menu_item_2, NULL);
menu->setPosition(Point(0, 0));
this->addChild(menu);
```

Enfin deux fonctions de rappel pour les événements de clic sur les boutons "Start" et "Exit". La fonction de rappel pour le bouton "Start" déclenche le passage à une nouvelle scène du jeu, appelée `LevelsMenu`, en utilisant un effet de transition. La fonction de rappel pour le bouton "Exit" ferme le jeu en utilisant la méthode `end()` de la classe `Director`.

```cpp
//la methode pour replacer la scenne de debut par la deuxieme scenne

void MainMenu::GotToLevelsMenu(cocos2d::Ref* pSender)
{
    auto scene = LevelsMenu::create();
    //replaccer la 1er scenne par un effet
    Director::getInstance()->replaceScene(TransitionFade::create(TRANSITION_TIME , scene));
}
//la methode pour Sortir du jeu

void MainMenu::menuCloseCallback(Ref* pSender)
{
    Director::getInstance()->end();
}
```

Figure 1 : Main Menu scene

III. Les niveaux du jeu :

III.1 Création de map par Logiciel Tiled :

Tiled est un éditeur de cartes pour jeux vedio qui permet de créer des cartes à défilement pour des jeux 2D. Le Logiciel utilise une grille régulière de tuile pour construire des cartes, ce qui permet de créer des cartes de manière rapide et efficace en utilisant un nombre limité de tuile.

Pour créer une carte il faut d'abord créer une tuile, en important des tuiles à partir d'images ou en dessinant sur la grille de tuile, dans notre cas on a importé une image de tuile.

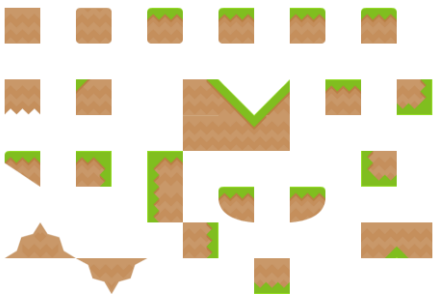

Figure2 : Image de tuile

Figure3 : Importation de Tuile sur Tiled

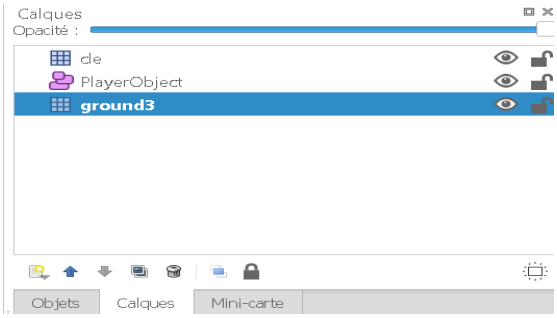

Figure4 : Liste des couches (Cle - ground3)

Après l'implémentation de Tuiles on place sur la grille les tuile pour créer une carte, on peut utiliser plusieurs couche (Layer) pour les obstacles pièges …Et ensuite on peut les sélectionner et les modifier sur cocos2d par la classe `TMXTiledMap` et la méthode `TMXTiledMap::getLayer ("Nom de la couche")`.

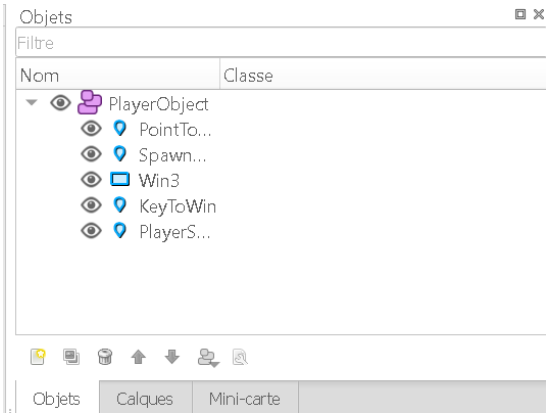

Figure5 : Liste des objets

On peut aussi ajouter des objets sur la carte pour les points de départ et d'arrivée, les objets sont de différents type : Point ou de forme géométrique différent (Rectangle – polygone), pour qu'ensuite les sélectionner et récupérer les données sur ces objets (x, y, width, height) par la classe `TMXObjectGroup` et ces méthodes :
`TMXObjectGroup::getObjectGroup("Nom du groupe d'objet")`, et
`TMXObjectGroup::getObject("Nom de l'objet")`.

Une fois que la carte est terminée, on l'importe dans le format XML (. TMX pour la carte du jeu et. TSX pour le jeu de tuile) ce qui le rend facile à utiliser dans le moteur de jeu Cocos2d-x-C++.

III.2 Mouvement du joueur :

Pour implémenter le mouvement par clavier il faut ajouter un écouteur d'événements pour les événements clavier et définit ses fonctions de rappel onKeyPressed et onKeyReleased sur respectivement `Level1::onKeyPressed` et `Level1::onKeyReleased`.

Ces fonctions de rappel seront appelées lorsqu'une touche est pressée ou relâchée. L'écouteur d'événements est ensuite ajouté au dispatcher d'événements avec une priorité de graphique de scène, de sorte qu'il recevra des événements avant les autres écouteurs d'événements.

Enfin, la fonction scheduleUpdate est appelée, ce qui indique au moteur d'appeler la fonction update sur cet objet à intervalle régulier.

```
////////////////////////////////////KEYBOARD IMPLEMENTATION/////////////////////////////////

auto eventListener = EventListenerKeyboard::create();

eventListener->onKeyPressed = CC_CALLBACK_2(Level1::onKeyPressed, this);
eventListener->onKeyReleased = CC_CALLBACK_2(Level1::onKeyReleased, this);

_eventDispatcher->addEventListenerWithSceneGraphPriority(eventListener, this);

this->scheduleUpdate();
////////////////////////////////////////////////////////////////////////////////////////////
```

Pour détecter la touche cliquée nous avons utilisé le conteneur `std ::map keys` qui stocke les éléments d'enum `EventKeyboard::KeyCode et chaque élément de ce dernier sera associé soit à true ou false.`

```
std::map<EventKeyboard::KeyCode, bool> keys;
void Level1::onKeyPressed(EventKeyboard::KeyCode keyCode , Event *event)
{
    keys[keyCode] = true;
}

void Level1::onKeyReleased(EventKeyboard::KeyCode keyCode, Event* event)
{
    keys[keyCode] = false;
}
```

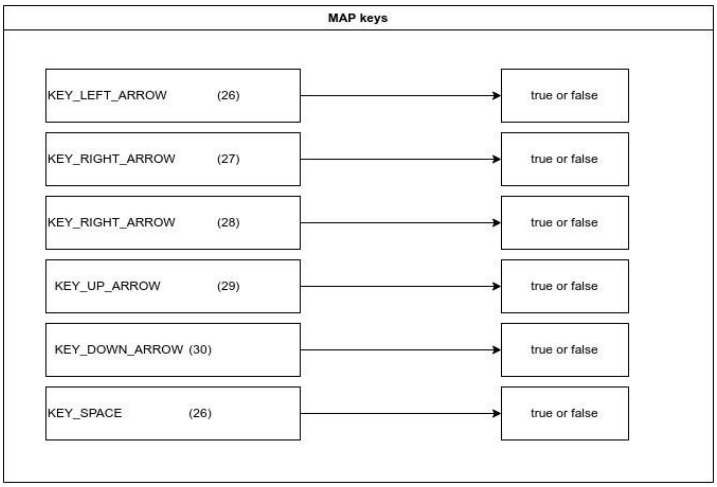

Figure 6 : diagramme représentant le conteneur std::map keys

- `Level1::update` : déplace le joueur selon la valeur les éléments du conteneur `std ::map` keys.

```
// la methode update de la class levvel1 se fait appeler a chaque frame du jeu
void Level1::update(float dt)
{
    float translate = 100.0 * dt;

    if (keys[EventKeyboard::KeyCode::KEY_LEFT_ARROW] == true ||
        keys[EventKeyboard::KeyCode::KEY_A] == true) { ... }
    else
    if(keys[EventKeyboard::KeyCode::KEY_RIGHT_ARROW] == true ||
      keys[EventKeyboard::KeyCode::KEY_D] == true) { ... }

    else
    if(keys[EventKeyboard::KeyCode::KEY_DOWN_ARROW] == true ||
      keys[EventKeyboard::KeyCode::KEY_S] == true) { ... }
    else
    if (keys[EventKeyboard::KeyCode::KEY_UP_ARROW] == true ||
        keys[EventKeyboard::KeyCode::KEY_Z] == true        ||
        keys[EventKeyboard::KeyCode::KEY_SPACE] == true) { ... }
```

III.3 Physiques du jeu

Cocos2d-x- contient deux moteurs du physique : Box2d et Chipmunk, et pour développer notre jeu nous avons choisis le moteur Chipmunk pour sa facilité d'utilisation.

Pour la création d'un corps physique il faut initialiser notre Scene par Chipmunk `initWithPhysics()`, puis créer le corps physique avec ces propriétés : masse, rotation, dynamique, restitution, friction, densité, avec la méthode `createBox()` en passant comme argument la taille du joueur avec la méthode `getContentSize()`.

```
//ajouter un physic corps dynamique et affecter par la graviter pour le "player"
    player_body = PhysicsBody::createBox(_player->getContentSize(), PhysicsMaterial(0.0f, 0.0f, 0.7f));
    player_body->setDynamic(true);
    player_body->setRotationEnable(false);              //player ne peut pas faire un mouvement de rotation
    player_body->addMass(10);
    _player->setPosition(x , y );
    _player->setPhysicsBody(player_body);
    this->addChild(_player);
```

Nous avons utilisé deux boucles for qui parcourt la grille en ajoutant un corps physique static a chaque tuile non nulle.

```
//ajouter un corp physique (Box) pur chaque Tile
    for (int x = 0; x < 70; x++)       //width of map
    {
        for (int y = 0; y < 32; y++)   //height of map
        {
            auto spriteTile = _background1->getTileAt(Vec2(x, y));
            if (spriteTile != NULL)
            {
                tilePhysics = PhysicsBody::createBox(Size(16, 16), PhysicsMaterial(0.0f , 0.0f , 2.0f));
                tilePhysics->setDynamic(false); //static is good enough for wall
                tilePhysics->setGravityEnable(false);
                tilePhysics->addMass(100);
                spriteTile->setPhysicsBody(tilePhysics);
            }
        }
```

Remarque : chaque tuile a un identifiant sous forme (x, y) qui permet de la sélectionner avec la méthode `getTileAt()`

Et dans la méthode `update` (), lorsqu' une des touches mentionnées dans la figure 6 est cliqué applique une impulsion sur le corps du joueur.

```
// la methode update de la class Level1 se fait appeler a chaque frame du jeu
void Level1::update(float dt)
{
    float translate = 100.0 * dt;

    if (keys[EventKeyboard::KeyCode::KEY_LEFT_ARROW] == true ||
        keys[EventKeyboard::KeyCode::KEY_A] == true)
    {
        _player->setScaleX(scale);                          //inverser playerSprite
        _player->getPhysicsBody()->setVelocityLimit(70);
        _player->getPhysicsBody()->applyImpulse(Vec2(-145, 0));

    }
    else
    if(keys[EventKeyboard::KeyCode::KEY_RIGHT_ARROW] == true ||
       keys[EventKeyboard::KeyCode::KEY_D] == true)
    {
        _player->setScaleX(scale *-1);                       //inverser playerSprite
        _player->getPhysicsBody()->setVelocityLimit(70);
        _player->getPhysicsBody()->applyImpulse(Vec2(145, 0));
    }

    else
    if(keys[EventKeyboard::KeyCode::KEY_DOWN_ARROW] == true ||
       keys[EventKeyboard::KeyCode::KEY_S] == true)
    {
        _player->getPhysicsBody()->applyImpulse( Vec2( 0 , -10) );
    }
    else
    if (keys[EventKeyboard::KeyCode::KEY_UP_ARROW] == true ||
        keys[EventKeyboard::KeyCode::KEY_Z] == true        ||
        keys[EventKeyboard::KeyCode::KEY_SPACE] == true)
    {
```

III.4 Détection des collisions

Dans notre jeu nous avons besoin détecter deux collisions : la collision du joueur avec les pièges et l'ennemis, et la collision du joueur avec la fin de chaque niveau.

✓ **Collision avec les pièges et les ennemies** :

Nous avons effectué une boucle à travers une grille de tuiles de taille 70x32 et vérifie si le Sprite du joueur intersecte (c'est-à-dire entre en collision) avec une tuile de l'eau ou la fin de niveau.

Pour chaque tuile de l'eau, il utilise la fonction intersectsRect pour vérifier si la bounding box (boîte englobante) de la tuile intersecte celle du joueur. Si c'est le cas, la scène en cours est remplacée par une scène de GameOver.

```
// collision de water et player et si intersectsRect() est true cad le player intersect avec water en remplace la scene avec GameOver Scene

    for (int x = 0; x < 70; x++)       //width of map
    {
        for (int y = 0; y < 32; y++)   //height of map
        {
            auto spriteTileWater = water->getTileAt(Vec2(x, y));
            if (spriteTileWater != NULL)
            {
                //si BoundingBox de player a intersecte BoundingBox de Layer Water en remplace la scene Level1 par GameOver Scene
                if (spriteTileWater->getBoundingBox().intersectsRect(_player->getBoundingBox())) {
                    auto scene = GameOver::create();
                    Director::getInstance()->replaceScene(scene);
                }
            }

        }
    }
```

✓ **Collision avec la fin de chaque niveau :**

Après la création d'un objet sous forme de rectangle à la fin de chaque niveau dans Tiled nous récupérons cet objet par la méthode `TMXObjectGroup::getObject("Nom de l'objet")` qui retourne un dictionnaire (std ::unordered_map) qui contient les informations sur l'objet.

Puis nous créons un `Rect` par le constructeur de la classe on utilise les informations l'objet (x, y, width, height).

```
//importer les coordonnés de l'objet "PlayerShowUpPoint" du fichier "Level1test.tmx" par la methode getObject() de la class TMXObjectGroup
    float x = playerShowUpPoint["x"].asFloat();
    float y = playerShowUpPoint["y"].asFloat();

    auto EndOfLevel1 = objects->getObject("Level1End");

    float width_end = EndOfLevel1["width"].asFloat();
    float height_end = EndOfLevel1["height"].asFloat();

    float x_end = EndOfLevel1["x"].asFloat();
    float y_end = EndOfLevel1["y"].asFloat();

    sizeEnd = Rect(x_end , y_end , width_end , height_end);
```

Et refait le même travaille de la collision avec Les pièges et les ennemies mais en replaçons la scene actuelle par Win scene.

III.5 Mouvement de la caméra :

Pour que la caméra suit le joueur nous avons utilisé l'action `Follow` qui suit le joueur selon un `Rect` qui est égale au boundingBox de Tiled map.

```
//Pour suivre le joueur selon un 'boundary' = mapsize
Rect mapsize = map_level1->getBoundingBox();
auto followplayer = Follow::create(_player , mapsize);
this->runAction(followplayer);
```

IV.5 Win et GameOver Scènes :

Win et GameOver class consistent à afficher une image de fond et un menu pour indiquer au joueur la réussite du niveau ou la perte.

Le menu de Win scene contient un buttons next qui replace la scene par LevelsMenu scene, tandis que dans le menu de GameOver contient deux buttons : Button restart qui replace la scene par LevelsMenu et le Button exit replace la scene par MainMenu scene.

Figure7 : GameOver scene

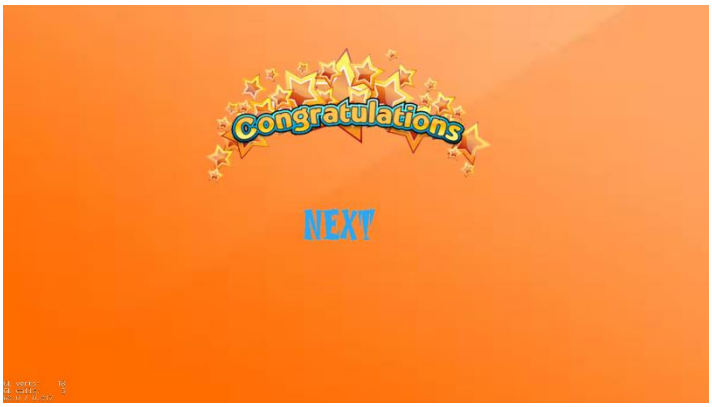

Figure 8 : Win Scene