Rapport Jeux 2D Pico Park

Réalisé par : *Ben Hammadi Sanae*

Diaz Khadija

Encadré par : PR. **IKRAM BEN ABDEL OUHAB**

PR. **ELAACHAK LOTFI**

Lien GitHub : https://github.com/sanaebenh/game-pico-park.git

À propos de cocos2d-x :

Le projet open source cocos2d-x est conçu pour être un moteur de jeu 2D multiplateforme permettant de créer des jeux 2D, des démos et d'autres applications mobiles graphiques/interactives. Il fonctionne sur OpenGL ES 1.1 et est écrit en langage C++, fournit une API C++(.Une API est un ensemble de définitions et de protocoles qui facilite la création et l'intégration de logiciels d'applications. API est un acronyme anglais qui signifie « Application Programming Interface », que l'on traduit par interface de programmation d'application.

Pourquoi choisir Cocos2d-x :

- API C++ moderne (veuillez-vous référer à la modernisation effectuée dans la **version 3.0**)
- Multiplateforme - ordinateur de bureau et mobile
- Capacité de tester et de déboguer votre jeu sur le bureau, puis de le pousser vers une cible mobile ou de bureau
- Une vaste API de fonctionnalités comprenant des sprites, des actions, des animations, des particules, des transitions, des minuteurs, des événements (tactile, clavier, accéléromètre, souris), du son, des E/S de fichier, de la persistance, des animations squelettiques, 3D

La Programmation orientée objet :

La programmation orientée objet (POO), ou programmation par objet, est un paradigme de programmation informatique. Elle consiste en la définition et l'interaction de briques logicielles appelées objets ; un objet représente un concept, une idée ou toute entité du monde physique.

La programmation par objet consiste à utiliser des techniques de programmation pour mettre en œuvre une conception basée sur les objets. Celle-ci peut être élaborée en utilisant des méthodologies de développement logiciel objet, dont la plus connue est le processus unifié (« Unified Software Development Process » en anglais), et exprimée à l'aide de langages de modélisation tels que le Unified Modeling Language (UML). **Le but :**

L'objectif principal de ce projet est de maitriser la programmation orientée objet par la mise en place d 'un jeu vidéo 2D, le jeu proposé s'appelle roller Splat, c'est un jeu qui a connu un grand succès dans les plateformes mobiles.

Réalisation :

Les classes utilisées :

Dans cette bibliothèque cocos2d-x pour la majorité de classe on trouve la déclaration de deux fichiers : le premier est de type « .h » qui sert à déclaration du classe, les méthodes … et le deuxième est de type « .cpp » qui sert à la définition et les implémentations.

Dans chaque classe, avant de commencer, on fait l'appel à des bibliothèques sous cette forme, #include "NomClass.h" car aura besoin des fonctions et des variables qui sont définies dans ces classes, c'est un principe fondamental pour la POO.

```
1
2   #include "AppDelegate.h"
3   #include "SplashScene.h"
```

1-AppDelegate.h / AppDelegate.cpp:

Cette classe est générée automatiquement par cocos2d , elle sert à : changer le nom de notre jeux , redimensionner la taille de notre fenêtre , création de notre scènes et l'appel de la scène suivante (SplashScene) .

Alors dans notre cas tout d'abord nous avons changé la taille :

```
USING_NS_CC;
  // change size
static cocos2d::Size designResolutionSize = cocos2d::Size(870, 680);
static cocos2d::Size smallResolutionSize = cocos2d::Size(870, 680);
static cocos2d::Size mediumResolutionSize = cocos2d::Size(1024, 768);
static cocos2d::Size largeResolutionSize = cocos2d::Size(2048, 1536);
```

Puis on a fait la déclaration du constructeur et de destructeur de notre classe qu'on va les utiliser tout de suite :

```
AppDelegate::AppDelegate()
{
}

AppDelegate::~AppDelegate()
{
#if USE_AUDIO_ENGINE
    AudioEngine::end();
#endif
}
```

Puis nous avons créé notre scène initiale (SplashScene) :

```
    // create a scene. it's an autorelease object
    auto scene = SplashScene::createScene();

    // run
    director->runWithScene(scene);

    return true;
```

Ainsi nous avons initialisé notre variable local (Director):

```
bool AppDelegate::applicationDidFinishLaunching() {
    // initialize director
    auto director = Director::getInstance();
    auto glview = director->getOpenGLView();
    if(!glview) {
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WIN32) || (CC_TARGET_PLATFORM == CC_PLATFORM_MAC) || (CC_TARGET_PLATFORM == CC_PLATFORM_LINUX)
        glview = GLViewImpl::createWithRect("RollerSplat", cocos2d::Rect(0, 0, designResolutionSize.width, designResolutionSize.height));
#else
        glview = GLViewImpl::create("RollerSplat");
#endif
        director->setOpenGLView(glview);
    }
```

2-Definitions.h :

Ce fichier sert à la définition de chaque variable qu'on va utiliser après ainsi la vitesse de transition entre les scènes et la vitesse de notre Sprite (player)

```
(Portée globale)
#ifndef __DEFINITIONS_H__
#define __DEFINITIONS_H__

#define DISPLAY_TIME_Splash_Scene 3  // la scene Splash_Scene va etre afficher durant 3 second
#define TRANSITION_TIME 1   // la vitesse de transition entre les diferents scenes est de 1
#define BALL_SPEED 0.1   // la vitesse du mouvement du ball
// les variables ayant la relation avec la position du ball
#define UP 100
#define DOWN 101
#define M 102
#define RIGHT 103
#define LEFT 104
#define XRIGHT 105
#define XUP 106
#define XXRIGHT 107
#define XXRIGHTM 108
#define LEFTM 109
```

```
// les variables ayant la relation avec les segments dessiné
#define SEGMENT_CODE_1 200
#define SEGMENT_CODE_2 201
#define SEGMENT_CODE_3 202
#define SEGMENT_CODE_4 203
#define SEGMENT_CODE_5 204
#define SEGMENT_CODE_6 205
#define SEGMENT_CODE_7 206
// les variables ayant la relation avec les stages du jeu
#define LEVEL_1 301
#define LEVEL_2 302
#define LEVEL_3 303

#endif // __DEFINITIONS_H__
```

3 – SplashScene.h / SplashScene.cpp

Dans cette class tout d'abord on fait la création de notre scène puis on fait la création d'un variable local(layer) et on l'ajout par la fonction « addChild »

```
Scene* SplashScene::createScene()
{
    // 'scene' is an autorelease object
    auto scene = Scene::create();

    // 'layer' is an autorelease object
    auto layer = SplashScene::create();

    // add layer as a child to scene
    scene->addChild(layer);

    // return the scene
    return scene;
```

Grace à la fonction scheduleOnce et DISPLAY_TIME notre scène va disparaitre après qlq second puis on passe à la scène suivante (MaineMenue).la position du background de notre scène est déterminé avec la fonction : setPosition ,Et bien sur la fonction addchild permet l'ajout de ce background, par contre le deuxième addchild permet d'ajouter un autre petit background sur le premier qui est aussi un sprite créé

```
    }
    //auto spriteFond = Sprite::create("bg.png");
    //SpriteFond->setAnchorpoint(0, 0);
   // SpriteFond->setAnchorpoint(0, 0);

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    this->scheduleOnce(CC_SCHEDULE_SELECTOR(SplashScene::GoToMainMenuScene), DISPLAY_TIME_SPLASH_SCENE);

    auto backgroundSprite = Sprite::create("CloseSelected.png");
    backgroundSprite->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));

    this->addChild(backgroundSprite);

    /*
    ///////////////////////////
    // 2. add a menu item with "X" image, which is clicked to quit the program
    //    you may modify it.

    // add a "close" icon to exit the progress. it's an autorelease object
    auto closeItem = MenuItemImage::create(
                                        "CloseNormal.png"
```

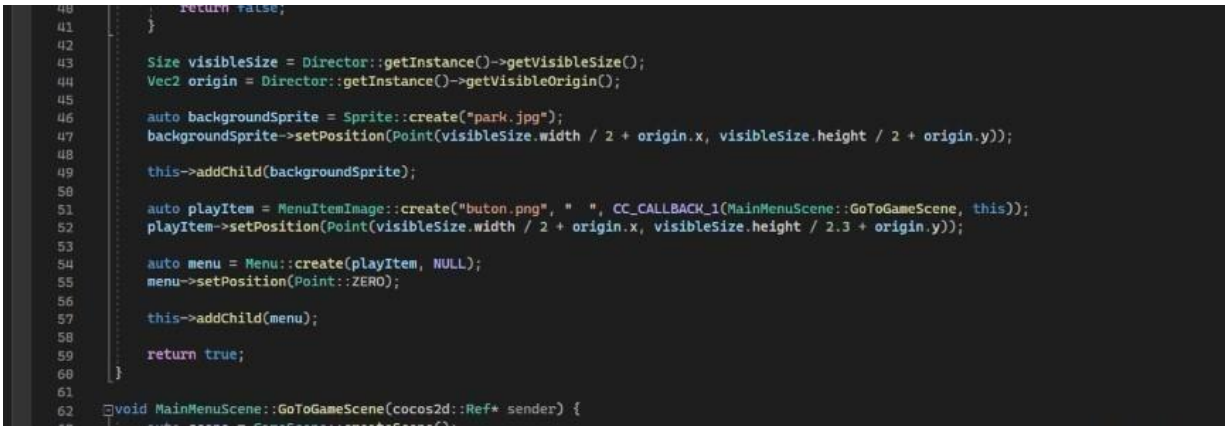
Et finalement nous avons créé une fct de type void permettant de passer à la scène suivante (MainMenue), et bien sûr il faut créer cette dernière avec la fonction createScene

```
void SplashScene::GoToMainMenuScene(float p)
{
    auto scene = MainMenuScene::createScene();

    Director::getInstance()->replaceScene(TransitionFade::create(TRANSITION_TIME, scene));
}
```

3 – MainMenu.h / MainMenu.cpp

Cette classe sert à insérer un menu pour note jeu ; ce menu contient un background et un bouton (Play) qu'on a inséré à l'aide des fonctions suivantes :



```
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    auto backgroundSprite = Sprite::create("park.jpg");
    backgroundSprite->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));

    this->addChild(backgroundSprite);

    auto playItem = MenuItemImage::create("buton.png", "  ", CC_CALLBACK_1(MainMenuScene::GoToGameScene, this));
    playItem->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2.3 + origin.y));

    auto menu = Menu::create(playItem, NULL);
    menu->setPosition(Point::ZERO);

    this->addChild(menu);

    return true;
}

void MainMenuScene::GoToGameScene(cocos2d::Ref* sender) {
```

Sprite sert à ajouter notre photo à la scène, float x et float y précise la position exacte de cette photo et finalement comme nous avons besoin d'un bouton cliquable affin de commencer notre jeu nous avons utilisé la fonction playItem qui sert de rendre une image cliquable. Et finalement nous avons remplacé cette scène avec la suivante (GameScène) avec la fonction déjà expliquer en dessus.

4-GameScene.h / GameScene.cpp :

Après la création de la scène, Layer et d'ajouter Layer en tant qu'enfant à la Scène et aussi l'ajout de notre arrière-plan qui va être valable pour tous les stages ; avec les mêmes fct déjà cité :

```

    auto visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    auto backgroundSprite = Sprite::create("lev.png");
    backgroundSprite->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));

    this->addChild(backgroundSprite);

    auto edgeBody = PhysicsBody::createEdgeBox(visibleSize, PHYSICSBODY_MATERIAL_DEFAULT, 3);

    auto edgeNode = Node::create();
    edgeNode->setPosition(Point(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));

    edgeNode->setPhysicsBody(edgeBody);

    this->addChild(edgeNode);
    player = new Player(this);

```

```
void GameScene::keyPressed(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event) {
    if (keyCode == EventKeyboard::KeyCode::KEY_UP_ARROW) {
        this->player->Up();

    }
    else  if (keyCode == EventKeyboard::KeyCode::KEY_DOWN_ARROW) {
        this->player->Down();

    }
    else  if (keyCode == EventKeyboard::KeyCode::KEY_LEFT_ARROW) {
        this->player->Left();

    }
    else  if (keyCode == EventKeyboard::KeyCode::KEY_RIGHT_ARROW) {
        this->player->Right();

    }

}
void GameScene::keyPressed(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event)
{
    if (keyCode == EventKeyboard::KeyCode::KEY_DOWN_ARROW)
    {
        this->player->DownKey();
        this->level->segmentColored(this->player->segmentCode);
    }
    if (keyCode == EventKeyboard::KeyCode::KEY_UP_ARROW)
    {
        this->player->UpKey();
        this->level->segmentColored(this->player->segmentCode);
    }
    if (keyCode == EventKeyboard::KeyCode::KEY_LEFT_ARROW)
    {
        this->player->LeftKey();
        this->level->segmentColored(this->player->segmentCode);
    }
```

Grâce à la fonction EventlistenerKeyboard on détecte les touches que l'utilisateur appuie cad les tests à faire sur player, les détails seront affichés sur la classe player, sans oublier aussi le segment qu'il faut le colorer quand le player passe par ce dernier.