Rapport du projet jeu 2d

(Pico Park)

Préparer par :

Rihab Touil

Sommaire :

Introduction :

Cocos2d-x est le moteur de jeu open source le plus populaire au monde. Cocos2d-x intègre deux langages de programmation principaux, C++ et Lua.

Installation :

- Installer python parce que Cocos2d est un framework de développement de jeux 2D open-source pour Python

- Installer cocos2d et l'extraire

- On change directory d'après l'invite de commande en cocos2d puis on effectue la commande suite python ./setup.py qui est utilisée pour installer des paquets Python à partir du code source .

- On crée un nouveau projet par la commande cocos avec les paramètres suivants :

  -new project name

-p nom de l’application

-l langage de programmation (cpp ou lua)

-d location pour générer le projet.

L’idée du jeu :

on a essayer de travailler sur le même concept du jeu PICO PARK ou chaque niveau se termine lorsque le joueur obtient les pièces d’or et les insérer dans un distributeur .

Menu :

Pour le menu on a créer une classe HelloWorld qui hérite de la classe Layer de Cocos2d. Une classe Layer représente une couche de contenu dans une scène Cocos2d. Une scène peut avoir plusieurs layer.

On trouve donc les méthodes suivantes :

static cocos2d::Scene* createScene() : Cette méthode statique permet de créer une scène qui contiendra la couche HelloWorld.

virtual bool init() : Cette méthode est appelée lors de l'initialisation de la couche.

void menuCloseCallback(cocos2d::Ref* pSender) : cette méthode est généralement appelée lorsque l'utilisateur souhaite fermer l'application ou quitter le jeu.

CREATE_FUNC(HelloWorld) : Cette macro permet de créer une instance de la classe HelloWorld.

void GoToGameScene(Ref *prender) : Cette méthode permet de passer à la scène du premier niveau.

```
#ifndef __HELLOWORLD_SCENE_H__
#define __HELLOWORLD_SCENE_H__

#include "cocos2d.h"

class HelloWorld : public cocos2d::Layer
{
public:
    static cocos2d::Scene* createScene();

    virtual bool init();

    //  selector callback
    void menuCloseCallback(cocos2d::Ref* pSender);

    // implement the "static create()" method manually
    CREATE_FUNC(HelloWorld);

    cocos2d::Sprite* mySprite;

    void GoToGameScene(Ref *prender);


};

#endif // __HELLOWORLD_SCENE_H__
```

definition :

```
Scene* HelloWorld::createScene()
{
    auto scene = Scene::create();


    auto layer = HelloWorld::create()


    scene->addChild(layer);


    return scene;
}
```

Créer une scène contenant un layer

```
bool HelloWorld::init()
{

    if (!Layer::init())
    {
        return false;
    }

    //---------------CC_CALLBACK_1 function called when the button is
    Size visibleSize = Director::getInstance()->getVisibleSize();
    Point origin = Director::getInstance()->getVisibleOrigin();

    //-------------------------l'image de plan d'arrier---------------

    mySprite = Sprite::create("img1.png");
    mySprite->setPosition(Point((visibleSize.width / 2) + origin.x, (
    this->addChild(mySprite);
    {
```

La fonction initialise le nouveau Layer et obtient la taille et l'origine de la fenêtre visible du moteur de jeu. Si la fonction échoue à initialiser le "Layer", elle renvoie "false".

Puis on crée des sprite pour ajouter les images

```
//-----------------creer shutdown -------------------
auto menu_item = MenuItemImage::create("quit.png", "quit.png", CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));
menu_item->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2) + origin.y));
//---------------MainMenu-----------------------------


auto play = MenuItemImage::create("play.png", "play.png", CC_CALLBACK_1(HelloWorld::GoToGameScene, this));
play->setPosition(Point((visibleSize.width / 2) + origin.x, (visibleSize.height / 2)+100 + origin.y));
```

On défini les boutons de jouer et quitter , on a créer un menu_item à l'aide de la classe "MenuItemImage" L'élément de menu est créé en utilisant deux images, "quit.png" avant le clique et "quit.png" après le clique , et en spécifiant une fonction de rappel, "menuCloseCallback", à appeler lorsque l'élément de menu est sélectionné. La position de l'élément de menu est définie au milieu de la fenêtre visible. Meme chose pour le bouton Play .

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

La définition des méthodes appeler lorsqu'on clique sur les boutons qu'on a créer.

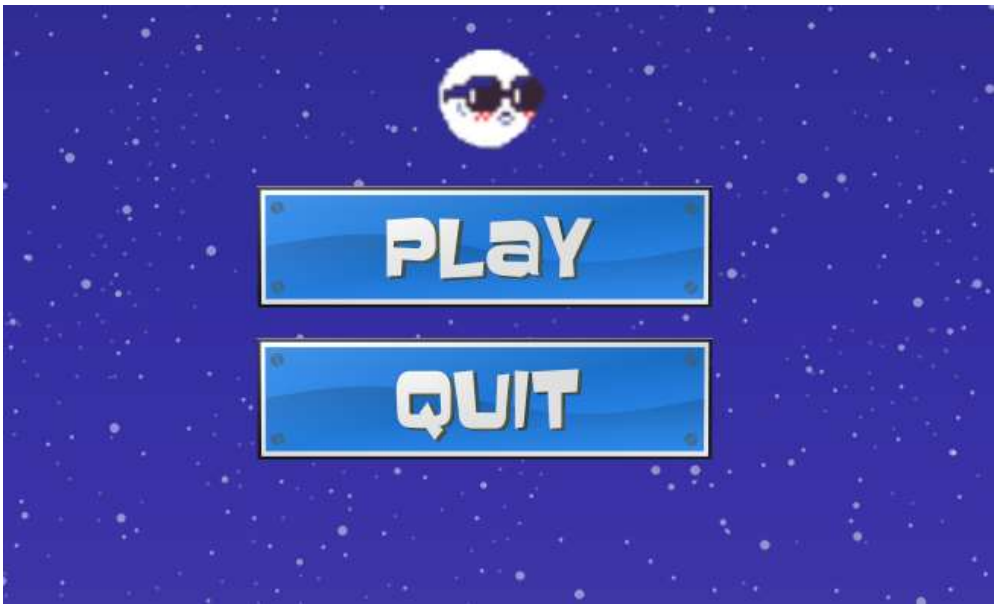
```
void HelloWorld::GoToGameScene(Ref* prender)
{
    auto scene = NewScene::createScene();
    Director::getInstance()->replaceScene(scene);
}

void HelloWorld::menuCloseCallback(Ref* pSender)
{
#if (CC_TARGET_PLATFORM == CC_PLATFORM_WP8) || (CC_TARGET_PLATFORM == CC_PLATFORM_WINRT)
    MessageBox("You pressed the close button. Windows Store Apps do not implement a close button.", "Alert");
    return;
#endif

    Director::getInstance()->end();

#if (CC_TARGET_PLATFORM == CC_PLATFORM_IOS)
    exit(0);
#endif
}
```

Play appelle la méthode GoToGameScene qui sert à créer un scène du premier niveau puis on remplace la scène actuelle avec la nouvelle .

Quit appelle la méthode menuCloseCallBack pour fermer la scène .

La classe Director est utilisée pour gérer les scènes et les transitions entre elles

On obtient donc :



Level 1 :

au premier niveau on a essayer de créer une simple map le joueur doit retenir la pièce d’or arriver au distributeur inserer la pière en cliquant sur entrer .

```
class NewScene : public cocos2d::Layer
{
public:
    static cocos2d::Scene* createScene();

    virtual bool init();

    // a selector callback
    void menuCloseCallback(cocos2d::Ref* pSender);

    // implement the "static create()" method manually


    cocos2d::Sprite* mySprite;
    cocos2d::Sprite* hero1;
    cocos2d::Sprite* coin1;

    cocos2d::Sprite* damage;
    cocos2d::Sprite* gr1;
    cocos2d::Sprite* gr2;
    cocos2d::Sprite* gr3;
    cocos2d::Sprite* tv;



    CREATE_FUNC(NewScene);
//private :
    //A world container is what your physics bodies are added to and where they are simulated
    //the interaction with PhysicsWorld object.
    cocos2d::PhysicsWorld* sceneWorld;
```

```
    CREATE_FUNC(NewScene);
/private :
    //A world container is what your physics bodies are added to and where they are simulate
    //the interaction with PhysicsWorld object.
    cocos2d::PhysicsWorld* sceneWorld;

    void SetPhysicsWorld(cocos2d::PhysicsWorld* World) { sceneWorld = World ; };
    bool NewScene::onContact(cocos2d::PhysicsContact& contact);
    int posX = 0;
    int posY = 0;
    int count = 0;
    void update(float dt);
    };

endif // __MAIN_MENU_H__
```

Au niveau du MainMenu.h on a déclarer les attribus pour les sprite ajouter dans le layer .

On a créer un monde de physique (objet "PhysicsWorld") c'est l'objet central qui gère les objets de physique dans une scène ,et nous aide à réaliser les differentes intéractions dans notre scène tel que les collisions la gravité les mouvement .

```
USING_NS_CC;

Scene* NewScene::createScene()
{

    auto scene = Scene::createWithPhysics();

   // scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
    scene->getPhysicsWorld()->setGravity(Vec2(0, -500));

    auto layer = NewScene::create();
    layer->SetPhysicsWorld(scene->getPhysicsWorld());

    scene->addChild(layer);


    return scene;
}
```

la fonction "onContact" est appelée lorsqu'il y a un contact entre deux objets de physique dans votre scène. Cette fonction prend un objet "PhysicsContact" en référence, qui contient des informations sur le contact entre les deux objets

```
bool NewScene::onContact(cocos2d::PhysicsContact& contact)
{
    PhysicsBody *bodyA = contact.getShapeA()->getBody();
    PhysicsBody *bodyB = contact.getShapeB()->getBody();

    if ((bodyA->getTag() == 1 && bodyB->getTag() == 2) ||
        (bodyA->getTag() == 2 && bodyB->getTag() == 1))
    {
        AudioEngine::play2d("Coin-Echo.mp3", false, 1.0f);
        coin1->removeFromParent();
        count++;

    }

    //next
    if ((bodyA->getTag() == 1 && bodyB->getTag() == 4) ||
```

Dans cette fonction "onContact", on récupére les deux corps de physique impliqués dans le contact en utilisant les méthodes "getShapeA" et "getShapeB" de l'objet "PhysicsContact". On a utilisé les méthodes "getTag" pour récupérer les étiquettes de chaque corps physique .

La première condition pour récuperer la pièce d'or le compteur count sert à verifier si la pièce sera récuperer ou non .

```
//next
if ((bodyA->getTag() == 1 && bodyB->getTag() == 4) ||
    (bodyA->getTag() == 4 && bodyB->getTag() == 1))
{
    auto  keyboardListener = EventListenerKeyboard::create();
    keyboardListener->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event)
    {
        if (keyCode == EventKeyboard::KeyCode::KEY_ENTER && count==1)
        {
            AudioEngine::play2d("coinsplash.mp3", false, 1.0f);
            auto scene = Level2::createScene();
            Director::getInstance()->replaceScene(scene);
        }

    };
    this->_eventDispatcher->addEventListenerWithSceneGraphPriority(keyboardListener, this);
    this->scheduleUpdate();
}
return true;
}
```

La deuxième conditon pour inserer les pièces dans le distributeur en cliquant sur entrer et avoir récupérer les pièces d'or pour passer au deuxième niveau .

```
auto  keyboardListener = EventListenerKeyboard::create();

keyboardListener->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event)
{
    switch (keyCode)
    {

    case EventKeyboard::KeyCode::KEY_UP_ARROW:
      posY += 4.0f;
       break;

    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
       posX -= 3.0f;
       break;

    case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
       posX += 3.0f;
       break;
    }

};

keyboardListener->onKeyReleased = [=](EventKeyboard::KeyCode keyCode, Event* event)
{
    switch (keyCode)
    {
```

```
this->_eventDispatcher->addEventListenerWithSceneGraphPriority(keyboardListener, this);
this->scheduleUpdate();
```

```
void NewScene::update(float dt)
{
    float newPosX = NewScene::hero1->getPositionX() + (posX);
    float newPosY = NewScene::hero1->getPositionY() + (posY);

    NewScene::hero1->setPosition(newPosX, newPosY);
    if (newPosY < 90)
    {
        AudioEngine::play2d("error.mp3", false, 1.0f);
        auto scene = GameOver::createScene();
        Director::getInstance()->replaceScene(scene);

    }
```

on a créer un gestionnaire d'évènements pour les touches du clavier et que tu l'enregistres avec la fonction addEventListenerWithSceneGraphPriority de l'objet _eventDispatcher. Les fonctions onKeyPressed et onKeyReleased seront appelées lorsqu'une touche du clavier sera enfoncée ou relâchée respectivement. La fonction scheduleUpdate indique que la méthode update de cet objet sera appelée à chaque frame

La fonction update est appelée à chaque frame et permet de mettre à jour l'état de l'objet en fonction du temps écoulé depuis la dernière frame (paramètre dt).

La fonction onKeyPressed est appelée lorsque l'utilisateur appuie sur une touche du clavier.

La fonction onKeyReleased est utilisée ici pour annuler l'action effectuée lorsque la touche est enfoncée.

On a donc pour résultat

<u>Level 2 et Level 3 :</u>

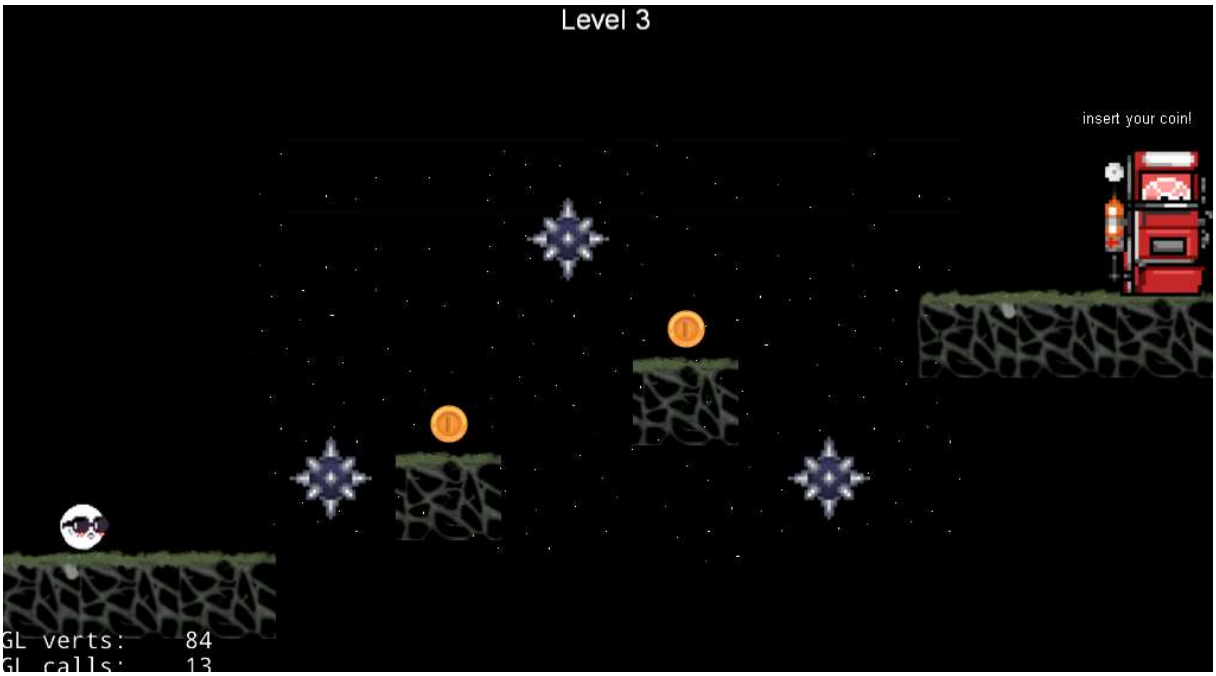
On a travailler avec les meme méthodes on a juster ajouter des obstacles pour augmenter la difficulté du stages

```
{
gr4 = Sprite::create("ball.png");

gr4->setPosition(Point(995, 380));
auto bgBody4 = PhysicsBody::createBox(gr4->getContentSize());
gr4->setPhysicsBody(bgBody4);

auto firstMvt = MoveTo::create(5.0, Point(10, 380));
auto secondMvt = MoveTo::create(5.0, Point(1000, 380));

auto seq = Sequence::create(firstMvt, secondMvt,nullptr);
auto re = RepeatForever::create(seq);

gr4->runAction(re);

bgBody4->setDynamic(false);
bgBody4->setContactTestBitmask(true);
bgBody4->setTag(3);

this->addChild(gr4);
```

On a créer deux mouvement de déplacement en deux point differents puis en ajoute les deux mouvemt séquenciellement pour avoir un effet de va et vient .

On obtient le résultat du Level2 :

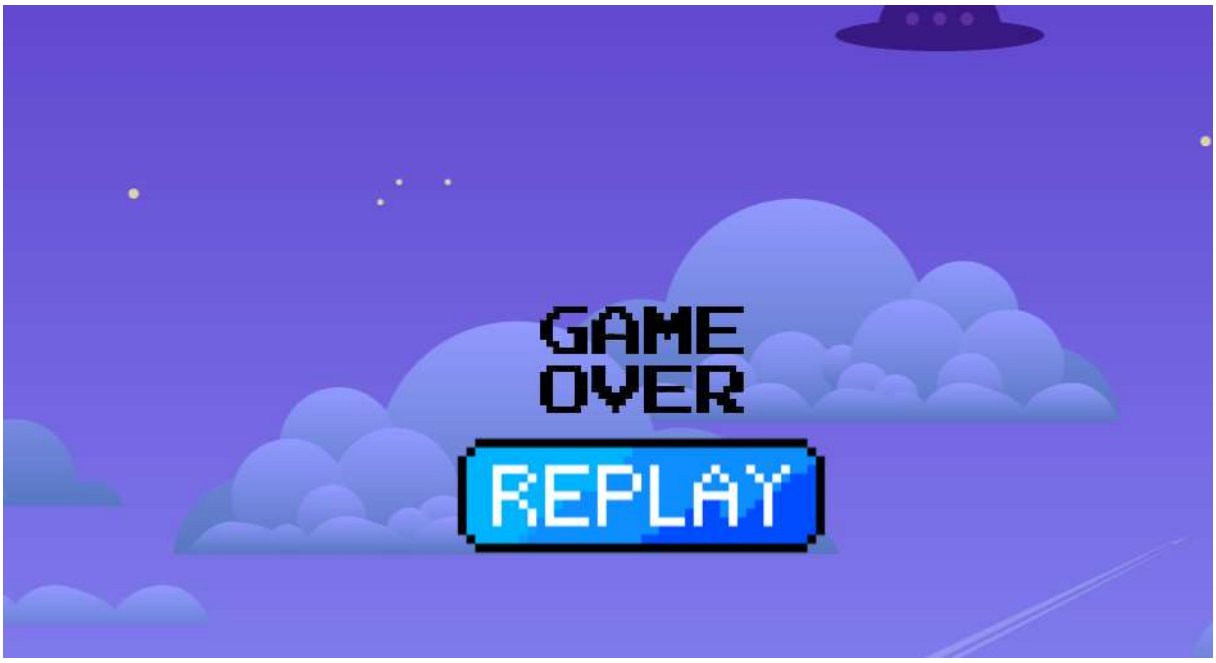
Et en Level 3 :



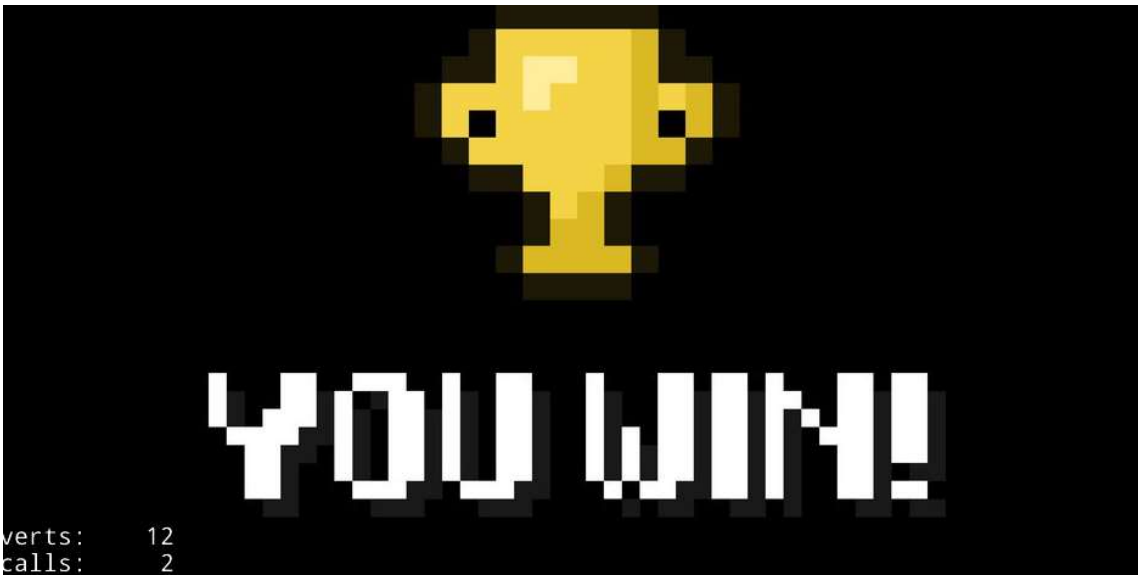
Et lors de la collision des obstacles avec le caractère la scène de game over sera afficher qui contient aussi un bouton pour rejouer .

```
//damage
if ((bodyA->getTag() == 1 && bodyB->getTag() == 3) ||
    (bodyA->getTag() == 3 && bodyB->getTag() == 1))
{
    AudioEngine::play2d("error.mp3", false, 1.0f);
    auto scene = GameOver::createScene();
    Director::getInstance()->replaceScene(scene);

}
//next
```

Le résultat :



Le 3 niveau nous mène lorsqu'on arrive au distributeur à la dernière scène qui déclare qu'on a gagné



<u>Ajouter le sons :</u>

On doit inclure la bibliothèque suivantes

```
#include "level2.h"
#include"level3.h"
#include"GameOver.h"
#include"AudioEngine.h"
```

Ajouter le son de format mp3 au ressources puis l'insérer

```
if ((bodyA->getTag() == 1 && bodyB->getTag() == 5) ||
    (bodyA->getTag() == 5 && bodyB->getTag() == 1))
{
    AudioEngine::play2d("Coin-Echo.mp3", false, 1.0f);
    coin2->removeFromParent();
    count++;

}
```

Conclusion :

Lors de la préparation de ce mini projet , j'ai pu pratiquer mes connaissances de la programmation orienté objet en c++ ainsi d'avoir une expérience d'utiliser engine cocos2d .