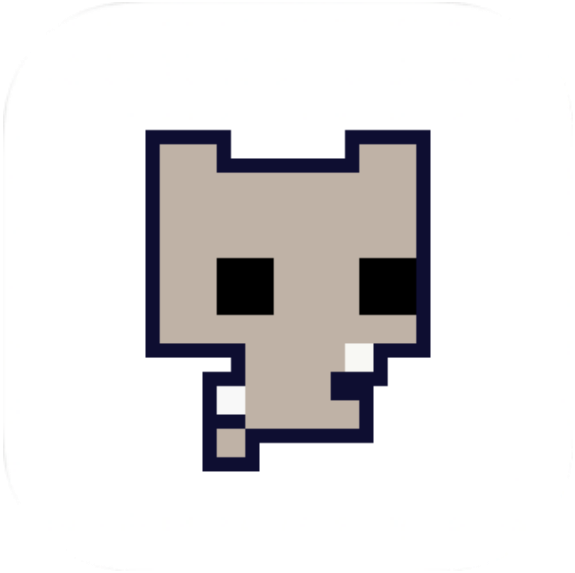
Université Abdelmalek-Essaadi Faculté des sciences
Et techniques-Tanger département génie informatique

Projet de module programmation oriente objet en CPP
Sujet : le jeux PicoPark

Lien github :



Réalisé par :

- Al-hamed Mohammed
- Achmal Chetouan Youssef

Encadré par :

- Ikram Benabdlouahab
- Lotfi El aachak

Table des matières

Table des Figures

1. Introduction :

Le domaine du jeux et un domaine qui est très large et il est un système par lesquelles joueurs s'engagent dans un conflit artificiel défini par des règles qui aboutit à un résultat quantifiable dans ce qui suit on va développer le jeux de RollerSplat avec cocos2d-x qui est un logiciel de création de jeux et aussi elle est une des bibliothèques logicielles de référence. En plus de permettre de créer des jeux en 2D pour les appareils mobiles Android, iOS et Windows Phone, elle compile sur Windows, Mac et Linux. La bibliothèque peut être utilisée pour le développement en C++, Javascript. Ce moteur de jeux vidéo est disponible en open source. Il dispose d'une grande communauté de développeurs qui le soutient et l'améliore chaque jour. Alors comment on va créer ce jeux ? Qui la logique suivit pour le crée ?
quelle sont les fonctions qui vont être utilise ?

1.1. Les objectifs

- La maitrise du langage CPP
- La maitrise de l'oriente objet
- L'utilisation du classe les objets les fonctions l'héritage.
- Apprendre comment programmer un jeu avec un logique bien

2. Description du jeu :

2.1. Définition de Pico Park :

Pico Park est un jeu multijoueur coopératif d'action-puzzle indépendant développé par TECOPARK. La version initiale de Pico Park pour Microsoft Windows était en 2016 via le détaillant de jeux vidéo Steam, proposant un jeu multijoueur local.

3. Les outils utiliser :

Pour réaliser ce projet on a utilisé plusieurs outils :

3.1. Cocos2d-X :

Cocos2d est un Framework libre en Python, permettant de développer des applications ou des jeux vidéo2.

Installation de Cocos2d-x :

• L'installation de Cocos2d-x nécessite l'installation de Python 2.7x. télécharger et installez Python version 2.7.15
• Après avoir installé Python, on ouvre la fenêtre PowerShell (ou ligne de commande) et on vérifie la version de python en tapant la commande python --version
• Téléchargez Cocos2d-x (cocos2d-x-4.0.zip) à partir de www.cocos2d-x.org et décompressez-le dans un dossier dans lequel vous voulez qu'il réside, par exemple C:\Dev\
• Configurez Cocos2d-x en exécutant setup.py dans un Shell d'alimentation depuis l'intérieur du dossier décompressé, par exemple C:\Dev\cocos2d-x-4.0\
• On configure d'abord certaines variables d'environnement, puis demande des chemins.
• Si Android Studio et Apache Ant sont installés sur votre système (dans mon cas, Ant est installé dans le dossier Dev) :
Pour la question ANT_ROOT, mettez C:\Dev\apache-ant-1.10.7\bin

Création du nouveau projet Cocos2d :

Pour faire de nouveaux projets, nous utiliserons la commande :
"cocos new 'ProJet name' -l Cpp" dans le powershell.

3.2. Cmake :

Cocos2d-x 4.0 utilise Cmake pour créer des fichiers de projet pour différentes plates-formes. Pour cette raison, nous avons besoin de Cmake installé sur notre machine.

Création Cmake dans notre fichier :

Pour ajouter Cmake a notre projet on utilise le command dans le fichier win32 :
« Cmake .. -G « Visual Studio 16 2019 » -Awin32 ».

3.3. Visual Studio:

Microsoft a une version gratuite de son IDE appelée « Visual Studio Community ». Il existe également une version open source gratuite appelée « Visual Studio Code ». Je vais aller avec la communauté car c'est ce qui est supporté par Cocos2d-x.

• Télécharger et exécutez le programme d'installation de la communauté Visual Studio.
• Depuis l'onglet Disponible, installez Visual Sudio 2017 (vous pouvez aussi opter pour 2019 !)
• Dans l'onglet Workloads, je recommande de sélectionner les unités « Python development », « Game development with C++ » et « Data storage and processing ». Vous en aurez probablement besoin plus tard.
• Appuyez sur Installer (ou Modifier) et suivez les instructions.

3.4. Langage Cpp :

C++ est le langue de programmation le plus utilisé par les développeurs, notamment en ce qui concerne les applications.il permet d'abord le développement sous plusieurs paradigmes : programmation génériques, procédurales et oriente objet.

Dans notre projet on a basé sur l'oriente objet en Cpp qui est un paradigme informatique consistant a définir et a faire interagir des objets grâce a différents technologies, notamment les langages de programmation (python, java, C++…).

3.5. Adobe Photoshop :

On a utilisée Adobe Photoshop pour crée les Sprite qu'on a ajouté a notre code.

4. Présentation du projet :

4.1. Logique de travail :

Pour chaque level le joueur doit finir un puzzle pour avoir la clé, en utilisant des collisions physiques, et par rectangle collision. En passant dans l'autre niveau par ouvrir le porte .

4.2. Les Classes :

On a créé plusieurs classes ms on peut les deviser on 4 :

4.2.1. Classe Des objets qu'on utilise dans notre scène :

Par exemple dans note jeux on a plusieurs objet a utilisé plusieurs fois :

- Player :
  Dans cette class on a créé notre joueur avec tout le fonctionnement qu'on va utiliser

```
USING_NS_CC;
Action* action;
OurPlayer::OurPlayer(cocos2d::Layer* layer) {

    Pico = Sprite::create("person.png");
    Pico->setScale(0.1);
    Pico->setPosition(Point(origin.x + 100, origin.y + 100));

    auto  Playerbody = PhysicsBody::createBox(Size(Pico->getContentSize().width-60, Pico->getContentSize().height), PhysicsMaterial(0, 0, 0));
    Playerbody->setCollisionBitmask(1);
    Playerbody->setContactTestBitmask(true);
    Playerbody->setRotationEnable(false);
    Pico->setPhysicsBody(Playerbody);
    layer->addChild(Pico, 1);
};

cocos2d::Vec2 OurPlayer::getposition() {
    return Pico->getPosition();
}

void  OurPlayer::setposition(float x, float y) {
    Pico->setPosition(Point(x, y));
}


cocos2d::Rect OurPlayer::getrect() {
    return Pico->getBoundingBox();
}



void OurPlayer::turnLeft(float z) {
    Pico->setRotation3D(Vec3(0,z,0));
}

void OurPlayer::turnRight()
{
    int x = Pico->getRotation3D().y ;
    if (x == 180)
    {
        Pico->setRotation3D(Vec3(0, 0, 0));
    }
```

- Box :

Dans cette class on a ajouté je carré que le joueur pushe dans le premier niveau avec tous ces caractéristiques et fonctionnement

```cpp
#include "box.h"

USING_NS_CC;

OurBox::OurBox(cocos2d::Layer* layer) {
    visibleSize = Director::getInstance()->getVisibleSize();
    origin = Director::getInstance()->getVisibleOrigin();
    Pico = Sprite::create("box.png");
    Pico->setScale(0.3);
    Pico->setPosition(Point(origin.x+300, origin.y+68));
    auto  Playerbody = PhysicsBody::createBox(Pico->getContentSize(), PhysicsMaterial(0, 0, 0));
    Playerbody->setCollisionBitmask(4);
    Playerbody->setContactTestBitmask(true);
    Playerbody->setDynamic(false);
    Pico->setPhysicsBody(Playerbody);
    layer->addChild(Pico, 1);
};

cocos2d::Vec2 OurBox::getposition() {
    return Pico->getPosition();
}

void  OurBox::setDynamic(bool dynamic) {
    Pico->getPhysicsBody()->setDynamic(dynamic);
}
void  OurBox::setposition(float x, float y) {
    Pico->setPosition(Point(x, y));
}
void OurBox::fall() {
    Pico->getPhysicsBody()->removeFromWorld();
}
void OurBox::removeFromParent() {
    Pico->setVisible(false);
}
cocos2d::Rect OurBox::getrect() {
    return Pico->getBoundingBox();
}
```

- Door :
  On a crée la porte avec ces deux statut si ouverte ou fermer et aussi des fonctionnement

```cpp
#include "door.h"

USING_NS_CC;

OurDoor::OurDoor(cocos2d::Layer* layer) {
    visibleSize = Director::getInstance()->getVisibleSize();
    auto s = Director::getInstance()->getWinSize();
    origin = Director::getInstance()->getVisibleOrigin();
    Door = Sprite::create("Door.png");
    Door->setScale(0.30);
    Door->setPosition(Point(s.width-100, origin.y + 90));
    OpenedDoor = Sprite::create("OpenedDoor.png");
    OpenedDoor->setScale(0.3);
    OpenedDoor->setPosition(Point(s.width - 100, origin.y+90));
    OpenedDoor->setVisible(false);

    layer->addChild(Door, 0);
    layer->addChild(OpenedDoor, 0);
};

cocos2d::Vec2 OurDoor::getposition() {
    return Door->getPosition();
}

void  OurDoor::setposition(float x, float y) {
    Door->setPosition(Point(x, y));
}

void  OurDoor::setposition2(float x, float y) {
    OpenedDoor->setPosition(Point(x, y));
}
void OurDoor::OpenDoor() {
    Door->setVisible(false);
    OpenedDoor->setVisible(true);
    exist = false;
}

cocos2d::Rect OurDoor::getrect() {
    return Door->getBoundingBox();
}
```

- key :
  La meme chose que les autre class des objet on a crée le clé avec ses « getters » et « setters » et des fonctionnement .

```cpp
#include "key.h"

USING_NS_CC;

OurKey::OurKey(cocos2d::Layer* layer) {
    visibleSize = Director::getInstance()->getVisibleSize();
    origin = Director::getInstance()->getVisibleOrigin();
    Pico = Sprite::create("key.png");
    Pico->setScale(0.25);
    Pico->setPosition(Point(origin.x +550, origin.y+150));
    layer->addChild(Pico, 1);
};

cocos2d::Vec2 OurKey::getposition() {
    return Pico->getPosition();
}

void  OurKey::setposition(float x, float y) {
    Pico->setPosition(Point(x, y));
}
void OurKey::removeFromParent() {
    Pico ->setVisible(false);
}
void OurKey::getkey(){
    auto action = MoveBy::create(20, Vec2(0,-1));
    Pico->runAction(action);
}
cocos2d::Rect OurKey::getrect() {
    return Pico->getBoundingBox();
}
```

4.2.2. Class Splashscene :

C’est la classe ou on a créé première scène qui s’affiche pour quelque second.

```cpp
bool SplashScene::init()
{
    // 1. super init first
    if ( !Scene::init() )
    {
        return false;
    }
    //creating auto of size of screen
    auto visibleSize = Director::getInstance()->getVisibleSize();
    //creating Vec2 of anchor point of screen
    Vec2 origin = Director::getInstance()->getVisibleOrigin();
    //executing method for switching this scene
    auto sprite1 = Sprite::create("HelloWorld.png");
    auto sprite2 = Sprite::create("logo.png");
    sprite1->setPosition(visibleSize.width-300, visibleSize.height/2);
    sprite1->setScale(1.1);
    sprite2->setPosition(visibleSize.width-700 , visibleSize.height / 2);
    sprite2->setScale(0.7);
    this->addChild(sprite1);
    this->addChild(sprite2);
    this->scheduleOnce(CC_SCHEDULE_SELECTOR(SplashScene::GoToMenuScene),1);
   return true;
}

 //method for switching scenes
void SplashScene::GoToMenuScene(float dt) {
    auto scene = MainMenu::createScene();
    Director::getInstance()->replaceScene(TransitionFade::create( 0.8, scene ));
}
```

4.2.3. Class Menu :

C’est la classe ou on a créé scène de menu dedans il y a le logo du jeu et un bouton pour commencer le jeu.

```
auto logo = Sprite::create("logo.png");
logo->setPosition(Vec2(visibleSize.width-15, visibleSize.height/1.4 ));
this->addChild(logo, -4);
logo->setScale(0.5);

auto playItem = MenuItemImage::create("button.png","pressed.png", CC_CALLBACK_1(MainMenu::GoToHelloWorldScene, this))
auto menu = Menu::create(playItem, NULL);
menu->alignItemsVerticallyWithPadding(visibleSize.height / 7);
this->addChild(menu);

return true;
```

4.2.4. Class des Niveaux :

- **Dans notre code c’est la class dont on ajoute des objets des class qu’on va utiliser :**

```
player = new OurPlayer(this);
key = new OurKey(this);
door = new OurDoor(this);
box = new OurBox(this);
```

- **Et aussi on a créé une camera :**

```
auto camera = Camera::create();
camera->setPositionZ(5);
camera->clearBackground();
this->addChild(camera);
```

- **Aussi on a changé « AnchorPoint » pour suivre notre joueur position comme ça camera va rester sure notre joueur avec son movement :**

```
Layer::setAnchorPoint(Vec2(player->getposition().x / v.width, player->getposition().y /v.height));
```

- **Dans le fichier header on a créé des méthode pour notre code qu’on a définie dans notre fichier cpp:**
  - On va commencer par la méthodes de création de la scène :

    La déclaration :

```
static cocos2d::Scene* createScene();
```

La définition :

```
Scene* MyWorld::createScene()
{
    auto scene = Scene::createWithPhysics();
    scene->getPhysicsWorld()->setDebugDrawMask(PhysicsWorld::DEBUGDRAW_ALL);
    scene->getPhysicsWorld()->setGravity(Vec2(0, -400));
    auto layer = MyWorld::create();
    scene->addChild(layer);
    return scene;
}
```

Ici on a créé notre scène dont on va ajouter tous notre élément du jeu, on a créé cette scène avec du physique pour ajouter les objets physiques et on l'a donné une gravité

- La méthode init :

La déclaration :

```
virtual bool init();
```

La définition :

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

Ici on voit si notre scène est créée si non on return « false » si oui on commence l'initialisation de notre variable

- Les méthodes de mouvement :

La déclaration :

```
void moveright(float dt);
void moveleft(float dt);
void movetop(float dt);
void movebot(float dt);
```

La définition :

```
void MyWorld::moveright(float dt) {
    Vec2 playerPos = player->getposition();
    player->setposition(playerPos.x + 80* dt, playerPos.y);
}
void MyWorld::moveleft(float dt) {
    Vec2 ballpos = player->getposition();
    player->setposition(ballpos.x - 80 * dt, ballpos.y);
}
void MyWorld::movebot(float dt) {
    Vec2 ballpos = player->getposition();
    player->setposition(ballpos.x, ballpos.y -80 * dt);
}
void MyWorld::movetop(float dt) {
    Vec2 ballpos = player->getposition();
    player->setposition(ballpos.x, ballpos.y + 160 * dt);
}
```

- Puis on a la méthode Update :

  Déclaration :

```
void update(float dt);
```

  C'est la méthode qu'on va utiliser pour savoir dans chaque moment où est notre « player » , pour détecter des collision sans « physicall body » et pour changer l'emplacement de la caméra .

  Définition :

```
void MyWorld::update(float dt) {

        Rect rect1 = player->getrect();
        Rect rect2 = key->getrect();
        Rect rect3 = door->getrect();
        Rect rect4 = box->getrect();
        Rect rect5 = wall->getBoundingBox();
        Rect rect6 = laser->getBoundingBox();
        auto v = Director::getInstance()->getWinSize();
        Layer::setAnchorPoint(Vec2(player->getposition().x / v.width, player->getposition().y /v

        if (rect1.intersectsRect(rect2))
        {
            key->setposition(player->getposition().x-10, player->getposition().y + 10);
            if (!keycollected)
            {
                cocos2d::AudioEngine::preload("audio/key.mp3");
                cocos2d::AudioEngine::play2d("audio/key.mp3", false, 0.3f);
                keycollected = true;
            }
```

- Finalement on a la méthode « onContactBegin () »

  Déclaration :

```
bool onContactBegin(cocos2d::PhysicsContact& contact);
```

Définition :

```
if ((1 == a->getCollisionBitmask() && 2 == b->getCollisionBitmask()) || (2 == a->getCollisionBitmask() && 1 == b->getCollisionBitmask()))
{
    ifSpacePressed = false;
    isOnGround = true;
}
if ((1 == a->getCollisionBitmask() && 3 == b->getCollisionBitmask()) || (3 == a->getCollisionBitmask() && 1 == b->getCollisionBitmask())) {
    if (ifDpressed)
    {
        this->unschedule(SEL_SCHEDULE(&MyWorld::moveright));
        player->setposition(player->getposition().x - 10, player->getposition().y);
    }
    else if (ifApressed)
    {
        this->unschedule(SEL_SCHEDULE(&MyWorld::moveleft));
        player->setposition(player->getposition().x + 10, player->getposition().y);
    }
}
if ((1 == a->getCollisionBitmask() && 4 == b->getCollisionBitmask()) || (4 == a->getCollisionBitmask() && 1 == b->getCollisionBitmask()))
{
    ispushing = true;
    ifSpacePressed = false;
    if (ifDpressed) {
        box->setposition(box->getposition().x + 4, box->getposition().y);

    }
    else if (ifApressed)
    {
        box->setposition(box->getposition().x - 4, box->getposition().y);
    }

}

if ((3 == a->getCollisionBitmask() && 4 == b->getCollisionBitmask()) || (4 == a->getCollisionBitmask() && 3 == b->getCollisionBitmask()))
{
    box->fall();
    box->removeFromParent();
```

Ici on pose une condition pour voir s'il y a une collision entre les deux objets si oui :

Le cas de collision entre joueur et box :

On voit dans quelle direction le joueur est dirigée :

Si droite on ajoute 4 px au position du box.

Si gauche on diminue 4px de la position x du box.

- **Pour déplacer le joueur dans notre scène à l'aide de notre « Keyboard » on a créé un « EventListenerKeyboard » en utilisant une fonction « onkeypressed »**
- **Pour arrêter le movement du joueur quand on arrête de toucher le bouton « Keyboard » on utilise une fonction « onkeyreleased »**

```
if (rect1.intersectsRect(rect3) && doorOpened == true && ifUpPressed==true) {
    ifUpPressed = false;
    auto scene = Lvl2::createScene();
    Director::getInstance()->replaceScene(TransitionFade::create(0.2, scene));
}
```

5. Interface du projet :

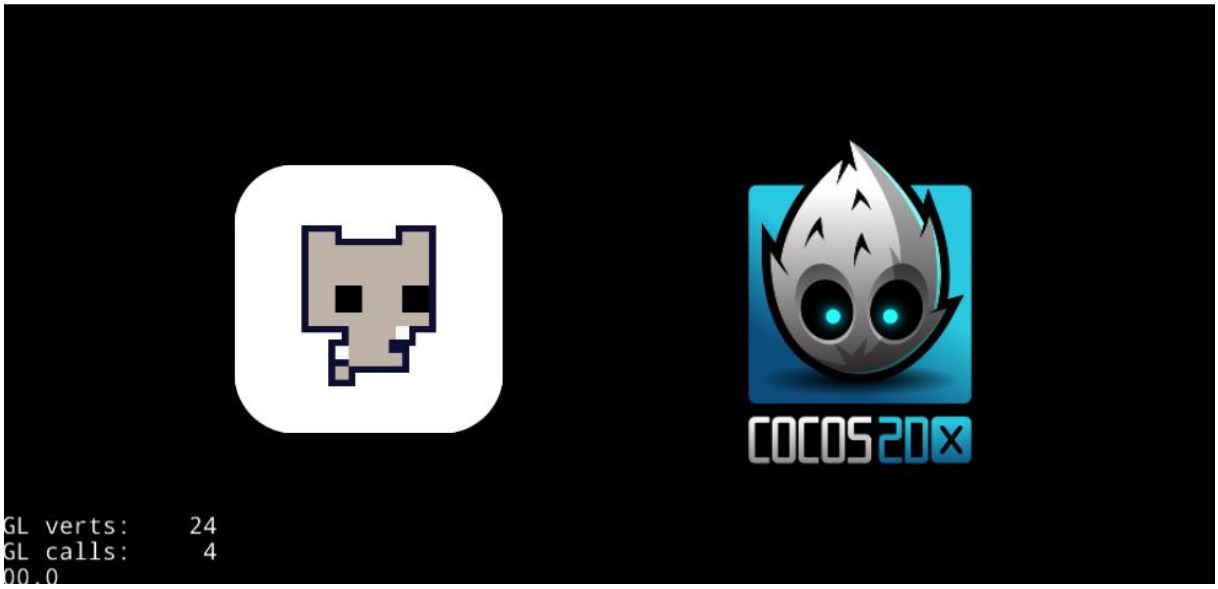


Figure 1 SplashScene

Figure 2Menu Scene

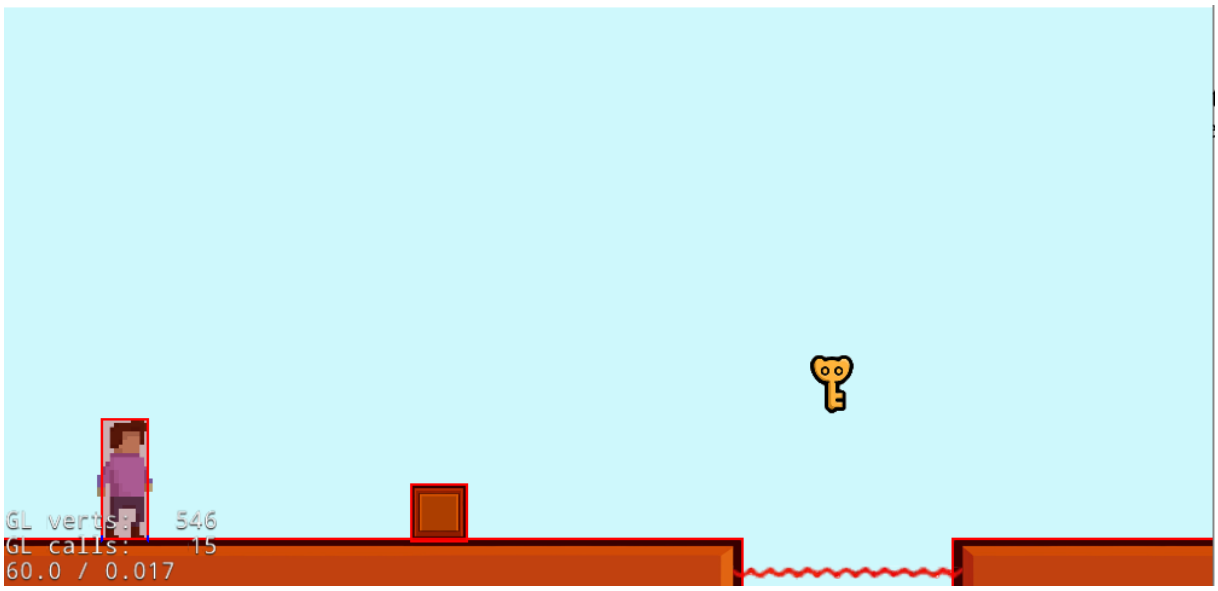


Figure 3 Level1 Scene

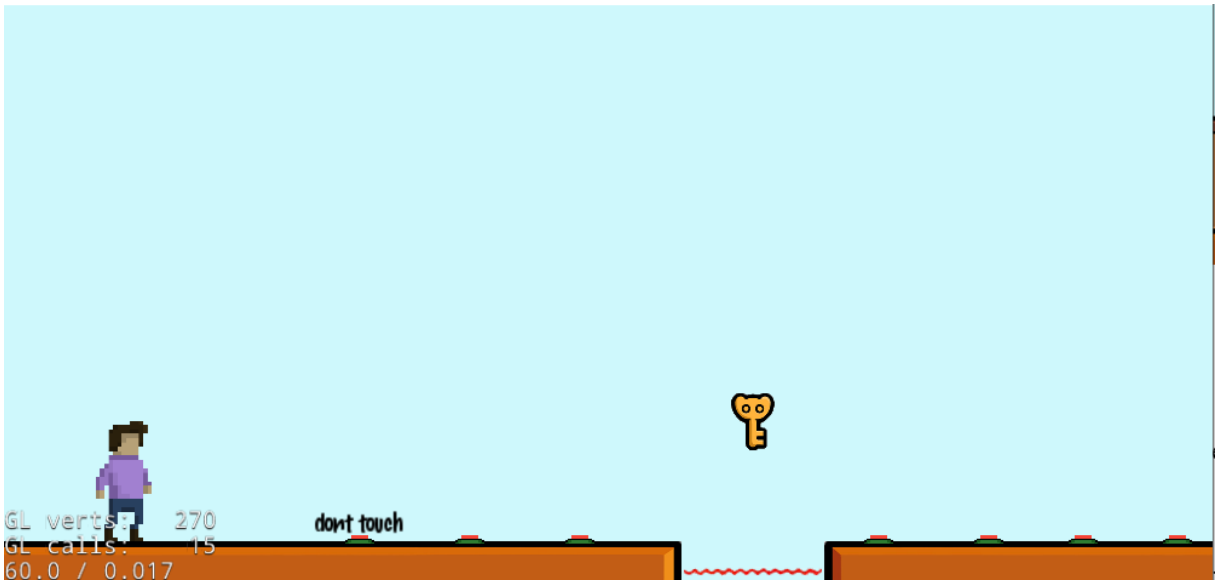


Figure 4 Level2 Scene

Figure 5 Level3 Scene

Figure 6 GameOver Scene

6. Conclusion

Apres avoir programmer ce jeux on a appris des nouveaux choses comme c'est quoi cocos2d-x ses différents fonctions qui sont déjà pris définit la logique des jeux et pour terminer c'était amusant de programme ce jeux .