COCOS2D-X
-CHAPTER 1-

SOULSEEK

목차

1. COCOS 2D-X?

1. COCOS 2D-X?

- 무료 라이센스로 제공되며 하나의 소스를 다양한 플랫폼에 맞추어 퍼블리싱 할 수 있다.
- 최초 **Cocos2d**로 시작되었는데 안드로이드 보다 아이폰 게임을 만들기 위한 엔진이 였고 여기서 **Cocos2d-x**가 나오면서 개발은 **C++**로 하면서 안드로이드와 **ios**로 멀티 플랫폼 개방 환경이 가능하게 되었다.
- **http://www.cocos2d-x.org/games** 으로 가서 이런 게임들이 만들어진 다는걸 알 수 있다.
- 여전히 **2D** 퍼즐게임에서는 **Cocos2d-x**를 많이 쓰고 있다.

Cocos2d – X 설치하기

- **http://www.cocos2d-x.org/download** 으로 이동한다.
  Cocos2d - x를 다운로드 받는다.
  원하는 폴더에 압축해제 한다.

파이썬 설치하기

- **https://www.python.org/downloads** 에서 **Python 2.x**버전을 다운로드 하고 원하는 곳에 압축해제 한다.
- 환경변수를 설정한다.
- 환경변수 **->**시스템변수
- **Path**항목에 파이썬 설치 폴더를 추가한다.
  cmd 창을 활성화 한다.
- **python -V**를 입력한다. 버전표시가 출력되면 정상.

1. COCOS 2D-X?

NDK 설치하기

- 공유해준 **NDK, ANT** 파일을 원하는 폴더에 압축 해제한다.

환경변수 설정

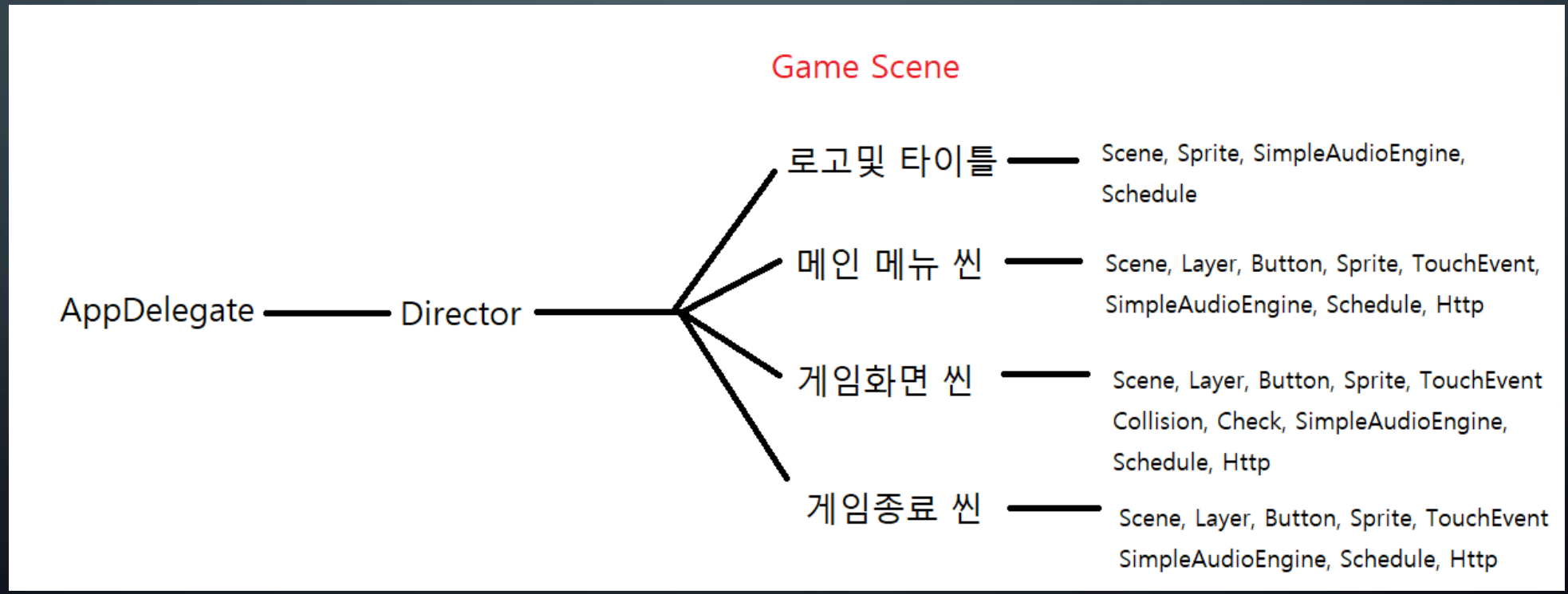
- **NDK**
  - **NDK_ROOT**을 사용자 변수에 추가해서 **NDK**경로를 추가해 준다.
- **Cocos2d-x**
  - **cmd** 창을 열고 **setup.py**파일을 드레그앤 드롭 하면 자동으로 환경설정을 해준다.
- **apache-ant**
  - **Path**에 **apache-ant**를 설치한 폴더를 추가해준다.

2. HELLOWORLD 분석

2. HELLOWORLD 분석

HelloWorld 프로젝트 생성

- **cmd**에서 **cocos2d-x** 설치 폴더로 이동해서 **cocos new HelloWorld -p com.SoulSeek.HelloWorld -l cpp -d ~/HelloWorld** 입력하여 프로젝트 생성
- **proj.win32**에서 솔루션을 실행하면 비쥬얼스튜디오 솔루션으로 실행된다.
- 추가적인 **H,CPP**파일들은 **class폴더 안에** 생성해야 된다.



2. HELLOWORLD 분석

기본 구성 요소

Director

- **Scene**들을 관리 하는 역활을 한다
- 각**Scene**들이 역활을 할것인지**. Sprite**나 사운드등은 어떻게 할것인지를 결정한다**.**
- 프로젝트의 기본 설정이 이루어 진다**.**

Scene

- 비쥬얼적인 구분이다
- 화면상의 한 장면을 말하며 게임에서 각각 타이틀 로비 인게임 결과 이렇게 큰 흐름의 구분이 되어지는 부분을 **Scene**라고 한다**.**

Layer

- **Scene**안에서 이루어지는 각각의 오브젝트들을 **Layer**라고 한다**.**
- **Sprite**를 그룹화 해서 움직이거나 동시에 효과를 줄수 있게 하기 위해 사용한다**.**
- 캐릭터와 머리위의 **HP**바를 같이 묶어서 그룹화한 것**.**

2. HELLOWORLD 분석

기본 구성 요소

Sprite

- **Layer**에 있는 오브젝트 구성원들을 하나하나를 말한다.
- 캐릭터, **HP**바

Action

- 구성된 **Scene, Layer, Sprite**를 움직이게 하는 것이다.
- 게임내의 화면제어를 제외한 **Scene**안에서의 움직임은 액션명령에 의해 정해진다.
- 스프라이트의 애니메이션과 동작처리를 위해 액션 클래스를 제공하고 있고 그것을 이용해서 이동, 회전, 점프, 크기변환등등을 제공하고 있으며 이 패턴들을 결합하여 사용할 수도 있고 반복해서 사용 할 수도 있다.
- **pMan->runAction(MoveBy::create(2, Vect2(50, 10)));**

2. HELLOWORLD 분석

기본 구성요소의 특징

Director

- **Scene**들을 관리 하는 역활을 한다
- 각**Scene**들이 역활을 할것인지**. Sprite**나 사운드등은 어떻게 할것인지를 결정한다**.**
- 프로젝트의 기본 설정이 이루어 진다**.**

Scene

- 비쥬얼적인 구분이다
- 화면상의 한 장면을 말하며 게임에서 각각 타이틀 로비 인게임 결과 이렇게 큰 흐름의 구분이 되어지는 부분을 **Scene**라고 한다**.**

Layer

- **Scene**안에서 이루어지는 각각의 오브젝트들을 **Layer**라고 한다**.**
- **Sprite**를 그룹화 해서 움직이거나 동시에 효과를 줄수 있게 하기 위해 사용한다**.**
- 캐릭터와 머리위의 **HP**바를 같이 묶어서 그룹화한 것**.**

2. HELLOWORLD 분석

Sprite

- **Layer**에 있는 오브젝트 구성원들을 하나하나를 말한다.
- 캐릭터, **HP**바

Action

- 구성된 **Scene, Layer, Sprite**를 움직이게 하는 것이다.
- 게임내의 화면제어를 제외한 **Scene**안에서의 움직임은 액션명령에 의해 정해진다.
- 스프라이트의 애니메이션과 동작처리를 위해 액션 클래스를 제공하고 있고 그것을 이용해서 이동, 회전, 점프, 크기변환등등을 제공하고 있으며 이 패턴들을 결합하여 사용할 수도 있고 반복해서 사용 할 수도 있다.
- **pMan->runAction(MoveBy::create(2, Vect2(50, 10)));**

좌표계

- **OpenGL**을 사용하므로 좌표계가 원점이 좌측하단이다. **y**값은 위로 **x**값은 오른쪽으로 증가한다.

Anchor

- 메뉴, 레이블. 스프라이트 등이 레이어나 장면에 포함될때 기준이 되는 부분이며 회전할때의 중심축이 된다.
- 앵커의 범위는 좌측상단부터 **(0,0)** 우측하단까지**(1,1)**의 값을 가진다.
- **sprite->setPosition(Vec2(240, 160));**

Input

- 여러가지 콘트롤러의 입력 이벤트를 처리한다.

2. HELLOWORLD 분석

자주쓰게될 자료형

- **cocos2d::Vec2 :**포지션이나 앵커값을 줄때 좌표값을 벡터값으로 주어지는 벡터**(**수학**)**타입**.**
- **ex) pMan->setPosition(Vec2(240, 160));**
- **pMan->setAnchorPoint(Vec2(0, 0));**
- **cocos2d::Size :** 크기설정이나 변환같은 곳에서 사용되어진다**. width, height**의 값을 객체로 가지고 있으며 사용할때 **Size.width, Size.height**로 사용된다**.**
- **cocos2d::Rect :2**차원 사각형을 정의하는데 사용된다**. x, y, width, height** 의 값을 가진다**.**
- **cocos2d::Color3B, cocos2d::Color4B :R,G,B**의 색깔값을 가진다**. Color4B**의 값은 거기에 **A**값이 추가된다**.**
- **cocos2d::Vector :std::vector**와 동일한 템플릿 클래스이다**.**

AppDeleGate Class

- 게임엔진 내부로 진입하기 위한 함수**. openGL** 화면 크기와 **Director,** 첫 시작화면의 설정을 지정과 외부요인으로 인해 앱이 잠시 중단되었을때의 처리를 기술한다**.**

```
bool AppDelegate::applicationDidFinishLaunching()
{
    //디렉터를 생성
    auto director = Director::getInstance();
    //창 크기와 창 이름
    glview = GLViewImpl::createWithRect("HelloWorld", cocos2d::Rect(0, 0, 1024, 768));
    //창을 생성해서 보여준다.
    director->setOpenGLView(glview);
    //FPS 체크할거라고 설정.
    director->setDisplayStats(true);
    //FPS 60프레임체크
    director->setAnimationInterval(1.0f / 60);
}
```

2. HELLOWORLD 분석

```
//해상도 조절
glview->setDesignResolutionSize(designResolutionSize.width,
                                designResolutionSize.height,
                                ResolutionPolicy::FIXED_HEIGHT);
```

- **EXACT_FIT :** 이미지를 꽉채우고 고정시키는것 기기 해상도에 맞에 이미지를 늘려서 꽉채우게된다.
- **NO_BORDER :** 테두리 없이 화면에 꽉채우는것 해상도를 벗어나는 위치에 있는 오브젝트들이 잘려서 보일수 있다.
- **SHOW_ALL : NO_BORDER**와 똑같지만 기기밖으로 나가는 것은 나가지 않게 화면안으로 넣어준다. 이렇게 하면 원래 원하던 오브젝트들의 배치나 간격이 틀려지게 된다.
- **FIXED_HEIGHT, FIXED_WIDTH :** 기기해상도에 사이즈를 맞춘 비율로 나타낸다. 조절된 값이 넘칠경우 게임화면이 잘리고 모자를 경우는 흑색처리가된다.

```
//입력된 값의 비율로 확대 축소되어 보여질것인지를 결정.
director->setContentScaleFactor();

//시작 할 Scene을 생성한다.(MainScene)
auto scene = HelloWorld::createScene();

//Scene을 실행한다.
director->runWithScene(scene);
```

2. HELLOWORLD 분석

- HelloWorld.h

```
#ifndef __HELLOWORLD_SCENE_H__
#define __HELLOWORLD_SCENE_H__

#include "cocos2d.h"

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

    virtual bool init();

    //콜백으로 지정할 함수.
    void menuCloseCallback(Ref* pSender);

    //"stactic node()" 메소드 구현.
    CREATE_FUNC(HelloWorld);
};

#endif // __HELLOWORLD_SCENE_H__
```

Scene을 생성하고 **Layer** 역할을 하는 **HelloWorld Class**를 생성한다.

2. HELLOWORLD 분석

- HelloWorld.cpp

```
//cocos2d의 클래스들을 사용할 것이다.
USING_NS_CC;

//Scene을 생성하고 HelloWorld를 Scene에 속하는 Layer로 만든다.
Scene* HelloWorld::createScene()
{
    //Scene을 생성한다.
    auto scene = Scene::create();
    //HelloWorld를 레이어로 생성한다.
    auto layer = HelloWorld::create();
    //생성한 레이어를 Scene의 자식으로 추가한다.
    scene->addChild(layer);
    //하나의 레이어를 가지는 Scene을 리턴한다.
    return scene;
}
```

2. HELLOWORLD 분석

```
bool HelloWorld::init()
{
    //상속받은 Layer클래스의 초기화가 실패하면 false 리턴
    if ( !Layer::init()
    {
        return false;
    }

    //실제 해상도와 화면의 크기를 가져온다.
    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();


    //Closebutton 역할을 할 이미지를 추가하고 클릭했을 때 호출 할 콜백 함수를 연결한다.
    auto closeItem = MenuItemImage::create(
    "CloseNormal.png",
    "CloseSelected.png",
    CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

    //CloseItem의 위치를 설정한다.
    closeItem->setPosition(Vec2(origin.x + visibleSize.width - closeItem->getContentSize().width / 2,
    origin.y + closeItem->getContentSize().height / 2));

    //Menu를 생성하고 CloseItem을 메뉴에 등록한다.
    auto menu = Menu::create(closeItem , NULL);
    menu->setPosition(Vec2::ZERO);
    this->addChild(menu, 1);
```

2. HELLOWORLD 분석

```
    //화면에 표시할 문자열을 설정한다.
    Auto label = Label::createWithTTF( " Hello World " ,  " fonts/Marker Felt.ttf " , 24);

    //문자열의 포지션을 설정한다.
    label->setPosition(Vec2(origin.x + visibleSize.width / 2,
    origin.y + visibleSize.height - label->getContentSize().height));

    //설정한 문자열을 레이어에 추가한다.
    this->addChild(label, 1);

    //Sprite를 생성한다.
    auto sprite = Sprite::create("HelloWorld.png");

    //Sprite의 위치를 설정한다.
    sprite->setPosition(Vec2(visibleSize.width / 2 + origin.x, visibleSize.height / 2 + origin.y));

    //Layer에 Sprite를 추가한다.
    this->addChild(sprite, 0);

    return true;
}
```

3. SPRITE CLASS

3. SPRITE CLASS

Sprite 생성

```
//스프라이트 생성 및 초기화를 한다.
Sprite* pMan = Sprite::create("grossiniaa.png");
//원하는 포지션에 위치한다.
pMan->setPosition(Vec2(110, 110));
//Z – Other를 지정해 주지 않는다.
addChild(pMan);
//Z - Other를 지정해준다.
//addChild(pMan, 1);
```

- 두 개 이상의 스프라이트가 존재할 때, 늦게 추가한 스프라이트가 상위에 그려진다.
- **Z – Other**를 지정해주면 지정해준 순서대로 그려진다.

Anchor

- **Sprite**들을 그룹화 하였을 때 상관관계에 따라 그려지는 위치의 중심을 잡아줘야 할 필요가 있다.

```
//하얀색 텍스처
Sprite* pHPSprite = Sprite::create("white-512x512.png");
//해당 스프라이트의 앵커포지션은 정가운데다.
pHPSprite->setAnchorPoint(Vec2(0.5f, 0.5f));
//위치포지션을 잡을때 앵커포지션의 위치를 기준으로 변한다음 그림이 그려진다고 보면된다.
pHPSprite->setPosition(pMan->getContentSize().width / 2, pMan->getContentSize().height + 15);
//각 프러퍼티들이 있다.
//컬러값을 줘서 색깔을 넣어준다.
pHPSprite->setColor(Color3B(0, 255, 0));
//원하는 크기만큼 그려주는 함수
pHPSprite->setTextureRect(Rect(0, 0, 50, 10));
//남자에게 HP바 추가
pMan->addChild(pHPSprite, 2);
```

3. SPRITE CLASS

Other Sprite Func

```
//투명도를 조절한다.
pHPSprite->setOpacity(255);

//이미지를 보여줄건지 숨길건지.
pHPSprite->setVisible(true);

//이미지를 회전시킨다.
pHPSprite->setRotation(90.0f);

//이미지를 x축으로 뒤집어 보여준다.
pHPSprite->setFilippedX(true);
```

4. LABLE CLASS

4. LABEL CLASS

시스템 폰트 사용

```
//시스템폰트는 LableTTF*를 사용.
//폰트가 있는 위치에 폰트를 이용해서 해당 글자를 적어준다.
LabelTTF* pLabel = LabelTTF::create("동해물과 백두산이 마르고 달토록 하느님이 보우하사
                                    우리나라만세", "fonts/arial.ttf", 30, Size(300, 300));

pLabel->setPosition(Vec2(240, 100));

//컬러값을 따로 줄수 있다.
pLabel->setColor(Color3B(0, 255, 0));
addChild(pLabel);
```

비트맵 폰트 사용

```
//BM폰트 비트맵 폰트라고 한다.
//비트맵 폰트를 만들어서 해당 글자를 적용한다.
LabelBMFont* pLabelBM = LabelBMFont::create("hello", "futura-48.fnt");
pLabelBM->setPosition(Vec2(240, 20));
addChild(pLabelBM);
```

5. INPUT EVENT

5. INPUT EVENT

Keyboard Event

```
//키보드 입력 리스너 생성
auto key_listener = EventListenerKeyboard::create();
//키보드는 CC_CALLBACK_2를 사용
//리스너에 키보드를 눌렀을때의 콜백을 등록
key_listener->onKeyPressed = CC_CALLBACK_2(InputEventScene::OnKeyPress, this);
//리스너에 키보드 눌렀다 놓았을때 콜백 등록
key_listener->onKeyReleased = CC_CALLBACK_2(InputEventScene::OnKeyUp, this);
//이벤트 디스패쳐에 이벤트리스너 등록
_eventDispatcher->addEventListenerWithSceneGraphPriority(key_listener, this);
```

Mouse Event

```
//마우스 입력 리스너 생성
auto mouse_listener = EventListenerMouse::create();
//마우스는 CC_CALLBACK_1 사용
//리스너에 마우스 상태에 따라 콜백을 등록
mouse_listener->onMouseDown = CC_CALLBACK_1(InputEventScene::OnMsDown, this);
mouse_listener->onMouseUp = CC_CALLBACK_1(InputEventScene::OnMsUp, this);
mouse_listener->onMouseMove = CC_CALLBACK_1(InputEventScene::OnMsMove, this);
mouse_listener->onMouseScroll = CC_CALLBACK_1(InputEventScene::OnMsScroll, this);
_eventDispatcher->addEventListenerWithSceneGraphPriority(mouse_listener, this);
```

5. INPUT EVENT

KeyBoard Callback

```
void InputEventScene::OnKeyPress(cocos2d::EventKeyboard::KeyCode keyCode, cocos2d::Event* event)
{
	switch (keyCode)
	{
		case EventKeyboard::KeyCode::KEY_UP_ARROW:
			UserUpMove(true);
		break;
		case EventKeyboard::KeyCode::KEY_DOWN_ARROW:
			UserDownMove(true);
		break;
		case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
			UserLeftMove(true);
		break;
		case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
			UserRightMove(true);
		break;
	}
}
```

Mouse Callback

```
void InputEventScene::OnMsDown(cocos2d::Event* event)
{

}
```