UNIVERSITÉ ABDELMALEK ESSAÂDI
FACULTÉ DES SCIENCES ET TECHNIQUES
TANGER

Rapport Game Cocos2dx :

Zahti Soukaina :

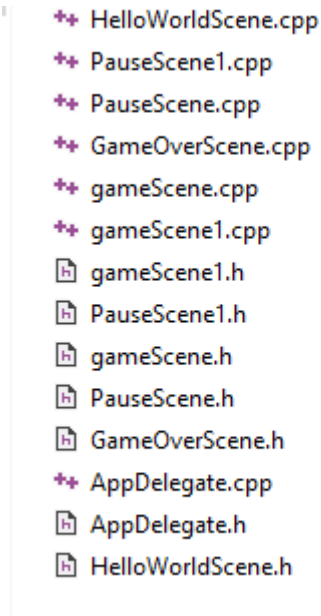
Coding process

Adding new scenes :

The easiest way to create new scenes is to duplicate the existing scenes provided by Cocos2d-x and modify them accordingly.

I added 4 different scenes :

The hello world scene, the pause scene, the game over scene and the game scene



implementing scenes

Without functionality, moving between scenes is not possible and they would be useless. Cocos2d-x provides great methods to move from one scene to another.

Coding the Main Menu scene

We Added the following code to the MainMenuScene.h file:

```
void GoToGameScene(Ref *pSender);
```

This is the declaration for the function that will be called when the player clicks on the play button from the Main Menu scene to replace it with the Game scene.

Then we Added the following code to the MainMenuScene.cpp file:

```
#include "GameScene.h"

void MainMenu::GoToGameScene(Ref *pSender) {    auto scene = GameScreen::createScene();
Director::getInstance()->replaceScene(scene); }
```

In the preceding code, the first line is used to include the Game scene so that the scene can be accessed. The GoToGameScene function first creates a local scene instance of GameScene and then replaces the current scene using the Cocos2d-x director.

Coding the Game scene

We added the following code to the GameScene.h file:

```
void GoToPauseScene(Ref *pSender);

void GoToGameOverScene(Ref *pSender);
```

In the preceding code, the first function declaration will be called when the player clicks on the pause button in the Game scene to push the Pause scene onto the stack. The second function declaration will be called when the player dies to replace the Game scene with the Game Over scene.

Add the following code to the GameScene.cpp file:

```
#include "PauseScene.h" #include "GameOverScene.h"

void GameScreen::GoToPauseScene(cocos2d::Ref *pSender) {    auto scene =
PauseMenu::createScene();       Director::getInstance()->pushScene(scene); }

void GameScreen::GoToGameOverScene(cocos2d::Ref *pSender) {    auto scene =
GameOver::createScene();       Director::getInstance()->replaceScene(scene); }
```

The first two lines in the preceding code snippet are used to include the Pause and Game Over scenes so that the scenes can be accessed. The GoToPauseScene function first creates a local scene instance of the Pause scene and then pushes it onto the stack. The GoToGameOverScene function first creates a local scene instance of the Game Over scene and then replaces the Game scene with it.

Coding the Pause scene

Add the following code to the PauseScene.h file:

```
void Resume(Ref *pSender); void GoToMainMenuScene(Ref *pSender);

void Retry(Ref *pSender);
```

In the preceding code snippet, the first function declaration will be called when the player clicks on the resume button from the Pause scene to pop the Pause scene off the stack. The second function declaration will be called when the player clicks on the main menu button from the Pause scene to replace the current scene with the Main Menu scene while popping all the scenes off the stack. The

third function declaration will be called when the player clicks on the retry button from the Pause scene to replace the current scene with the Game scene while popping all the scenes off the stack.

The GoToMainMenuScene function first creates a local scene instance of the Main Menu scene, then it pops the current scene off the stack and the Game scene is replaced with the Main Menu scene. The Retry function first creates a local scene instance of the Game scene, then it pops the current scene off the stack and the current Game scene is replaced with the new Game scene (restarts the game)

Coding Game Over scene

We Add the following code to the GameOverScene.h file:

void GoToGameScene(Ref *pSender); void GoToMainMenuScene(Ref *pSender);

In the preceding code, the first function declaration will be called when the player clicks on the retry button from the Game Over scene to replace the current scene with the Game scene. The second function declaration will be called when the player clicks on the main menu button from the Game Over scene to replace the current scene with the Main Menu scene.

Then we implement the functions by Adding the following code to the GameOverScene.cpp file:

```
#include "GameScene.h" #include "MainMenuScene.h"

void GameOver::GoToGameScene(cocos2d::Ref *pSender) {    auto scene =
GameScreen::createScene();       Director::getInstance()->replaceScene(scene); }

void GameOver::GoToMainMenuScene(cocos2d::Ref *pSender) {    auto scene =
MainMenu::createScene();       Director::getInstance()->replaceScene(scene); }
```

The first two lines in the preceding code snippet are used to include the Game and Main Menu scenes so that the scenes can be accessed. The GoToGameScene function first creates a local scene instance of the Game scene and then replaces the Game Over scene with it. The GoToMainMenuScene function first creates a local scene instance of the Main Menu scene and then replaces the Game Over scene with it.

Adding Game Menus

Menus consist of menu items, for example, a menu item image that will be used for this game. There are several items that are provided by Cocos2d-x to construct menus

Menus are a collection of items that are organized to form the structured buttons and can be used for a game's functionality, such as navigation. Our game will use menus for the following use cases: • In the Main Menu scene: ° The play button, which starts the game • In the Game scene: ° The pause button, which pauses the game • In the Pause scene: ° The resume button, which resumes the game ° The retry button, which restarts the game ° The main menu button, which takes you to the main menu • In the Game Over scene: ° The retry button, which restarts the game ° The main menu button, which takes you to the main menu Each scene will require its own menus along with its own menu items. All the menus will be declared and constructed within the init() method, but if the requirement for manipulating the menu or menu items arises, it would be best to declare them within the header so that they can be accessed outside the init() method.

Example :

Add the following code to the init() method:

```
auto menuTitle =  MenuItemImage::create("MainMenuScreen/Game_Title.png",
"MainMenuScreen/Game_Title.png");

auto playItem =  MenuItemImage::create("MainMenuScreen/Play_Button.png",
"MainMenuScreen/Play_Button(Click).png",  CC_CALLBACK_1(MainMenu::GoToGameScene, this));

auto menu = Menu::create(menuTitle, playItem, NULL); menu-
>alignItemsVerticallyWithPadding(visibleSize.height / 4);

this->addChild(menu);
```

Menu item images have three states:

normal (not being pressed),

pressed (user is tapping it),

and disabled (item has been disabled). The first statement in the preceding code snippet creates a menu title using the menu item image with the following parameters:

• The default image that has to be displayed in the title

• The image that has to be displayed when the title is tapped on (as the title cannot be tapped on, it is the same as the default image to give the illusion of a static item) The second statement creates an image item with the following parameters:

• The default image that has to be displayed on the button

• The image that has to be displayed when the button is being tapped on

• The function that has to be called when the button is tapped on, which will take the player to the Game scene.

We went through the same process with the other scenes.

The options i developped :

In the game scene, i implemented the code for the main game.

There's a single player, i gave it the option to jump when i press the space key or the up arrow

Key by implementing the following code to avoid collision with a ball heading towards it :

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


  eventListener->onKeyPressed = [=](EventKeyboard::KeyCode keyCode, Event* event)
  {
      // Set the velocity of the sprite based on the key that was pressed
      Vec2 velocity;
      switch (keyCode)
```

```
    {
    case EventKeyboard::KeyCode::KEY_LEFT_ARROW:
    case EventKeyboard::KeyCode::KEY_A:
        velocity = Vec2(-50, 0);
        mySprite->setScaleX(-1); // Flip the sprite horizontally
        break;
    case EventKeyboard::KeyCode::KEY_RIGHT_ARROW:
    case EventKeyboard::KeyCode::KEY_D:
        velocity = Vec2(50, 0);
        mySprite->setScaleX(1); // Un-flip the sprite horizontally
        break;
    case EventKeyboard::KeyCode::KEY_UP_ARROW:
    case EventKeyboard::KeyCode::KEY_SPACE:

        velocity = Vec2(0, 100);
        break;
    }

    // Apply the velocity to the sprite's physics body
    mySprite->getPhysicsBody()->setVelocity(velocity);

};
eventListener->onKeyReleased = [=](EventKeyboard::KeyCode keyCode, Event* event)
{
    Vec2 velocity = Vec2(0, 0);
    // When the left or right arrow key is released, stop the sprite
    switch (keyCode)
    {
    case EventKeyboard::KeyCode::KEY_UP_ARROW:
    case EventKeyboard::KeyCode::KEY_SPACE:
        mySprite->getPhysicsBody()->setVelocity(velocity);
```

```
    }
    };
    // Don't forget to add the event listener to the event dispatcher
    this->_eventDispatcher->addEventListenerWithSceneGraphPriority(eventListener, mySprite);

    auto contactListener = EventListenerPhysicsContact::create();
    contactListener->onContactBegin = CC_CALLBACK_1(GameScene::onContactBegin, this);
    this->getEventDispatcher()->addEventListenerWithSceneGraphPriority(contactListener, this);
```

If the player succeded at avoiding the ball for a certain amount of times, we go to the next level developped by my partner.

And If the player collides with an ball, then the game will transition to the Game Over scene.

I simply added a function to move between the two scenes like i explained before.

In oder to use the code mentiond to move the sprite and to detect collision, first i created a physics body for the ball, which will be a circle for the purpose of this game

Then i set the body up for collision detection .

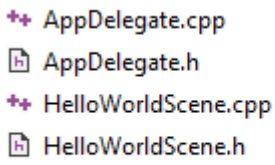
and The finaly i assigned the body to the player's sprite.

Chahid Oumama

Le but de ce rapport est de décrire le processus de création d'un jeu vidéo sur Cocos2dx, une version améliorée et étendue de Cocos2d, un framework de développement de jeux en 2D. J’ai utilisé Cocos2dx pour créer un jeu de type " defense against fish ", où le joueur doit se déplacer pour qu'il se protégé contre les poissons ennemis.

Phase de conception :

Au début du projet Pour développer le jeu, j’ai utilisé l'IDE (environnement de développement intégré) Visual Studio et la bibliothèque Cocos2dx pour la programmation en C++. J’ai commencé par mettre en place la structure de base du jeu, en créant les classes des différents éléments du jeu et en implémentant les mécanismes de gameplay,



Pour coder Mon Projet j'ai commencé par créer une scene dans « HelloWorldScene »

```
#include "HelloWorldScene.h"
#include "cocos2d.h"
#include"AppDelegate.h"
using namespace cocos2d;
Scene* HelloWorld::createScene()
```

Et Apres j'avais un seul joueur, que je lui ai donné la possibilité de se déplacer a droite quand la souris et a droite et de se déplacer a gauche quand la souris et à gauche pour éviter une collision avec l'ennemis poison quand il tombe vers lui

Le code que j'ai utilisé pour le déplacement est :

```
void HelloWorld::movePlayer(Touch* touche, Event* evento)
{
	auto positionTouch = touche->getLocation();
	if (_classic->getBoundingBox().containsPoint(positionTouch)) {
		_classic->setPositionX(positionTouch.x);
	}

}
```

Apres j'ai créé mais ennemis avec ce code mais j'avais un grand problème

```
void HelloWorld::monsterfish(float ct)
{
	auto director = Director::getInstance();
	auto size = director->getWinSize();
	Sprite* fish = nullptr;
	for (int i = 0; i < 3; i++)
	{
		fish = Sprite::create("image/poi.png");
		fish->setAnchorPoint(Vec2::ZERO);
		fish->setPosition(CCRANDOM_0_1() * size.width, size.height);
		initializePhysics(fish);
		fish->getPhysicsBody()->setVelocity(Vect(0, ((CCRANDOM_0_1() + 0.2f) * -250)));
		_fishs.pushBack(fish);
		this->addChild(fish, 1);


	}
}
```

Les cercle des poissons ne s'affiche pas j'ai cherché et j'ai trouver que je dois ajouter « Shedule_selector » dans init mais malgré ça mon code me donne une erreur quand je tape :

```
schedule(schedule_selector(HelloWord :: monsterfish)5.0f)
```

L'erreur était dans Schedule_selector 'not defined' j'ai ajouté 'float ct' dans void mais rien n'a changer j'ai ajouté `using namespace cocos2d;`
Mais rien n'a changer. Cette erreur ma pris 5 jours de recherche j'ai fait tout j'ai ajouté `This->` mais ça ne change rien et après 5 jours j'ai Trouver que je peux l'écrire a cette façon

```
this->schedule(CC_SCHEDULE_SELECTOR(HelloWorld::monsterfish), 5.0f);
```

et ça a marcher

après j'ai créé sprite , un cercle autour des poissons et du joueur car le concept ils sont dans l'eau

```
void HelloWorld::initializePhysics(cocos2d::Sprite* sprite)
{
        auto body = PhysicsBody::createCircle(sprite->getContentSize().width / 2);
        body->setContactTestBitmask(true);
        body->setDynamic(true);
        sprite->setPhysicsBody(body);
}
```

Ensuite J'avais un autre problème des photos que je vais utiliser j'ai choisie mon image de poisson mais l'image elle était très grande ensuite j'ai trouvé que d' à travers un site des images je peux contrôler size d'une image

Et Pour que Mon code puisse Travailler j'avais besoin d'ajouter toutes les fonctions que j'avais ajouté dans « HelloWorldScene.h »

```
  CREATE_FUNC(HelloWorld);
private:
    cocos2d::Sprite* _classic;
    cocos2d::Vector<cocos2d::Sprite*> _fishs;
    void initializeTouch();
    void monsterfish(float dt);
    void initializePhysics(cocos2d::Sprite* sprite);
    void movePlayer(cocos2d::Touch* toque, cocos2d::Event* event);
```

Une fois que j'ai terminé le développement du jeu, j'ai effectué des tests et débogué le code afin de corriger les erreurs et de s'assurer que le jeu fonctionnait correctement. J'ai également effectué des tests de performance afin de vérifier que le jeu fonctionnait de manière fluide et sans ralentissements indésirables.

Une fois que nous avons terminé le test et le débogage, nous avons publié le jeu sur GitHub . Nous avons créé pour le jeu une vidéo de démonstration des niveaux

Remerciement

Chère Professeure Ikram Ben Abdel Ouahab,

Cher Professeur Lotfi Laachak ,

Nous Tenons à vous remercier sincèrement pour votre enseignement et votre guidance cette année. Votre passion pour votre matière et votre dévouement à aider les étudiants à réussir ont été une inspiration pour nous.

Nous sommes très reconnaissantes pour tout le temps et l'énergie que vous avez consacrés à notre classe et à chacun de nous. Votre patience et votre compréhension ont été très appréciées.

Nous sommes persuadés que votre enseignement nous aidera à réussir dans nos études futures et à poursuivre nos rêves.

Encore une fois, merci pour tout ce que vous avez fait pour nous et pour notre classe cette année.

Cordialement,

Chahid Oumama , Zahti Soukaina